RealView® Development Suite

Version 3.1

CodeWarrior IDE Guide

ARM®

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.
ARM DUI 0065G

RealView Development Suite
CodeWarrior IDE Guide

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
October 1999	A	Non-Confidential	Release 1.0 (ADS)
March 2000	B	Non-Confidential	Release 1.0.1 (ADS)
November 2000	C	Non-Confidential	Release 1.1 (ADS)
November 2001	D	Non-Confidential	Release 1.2 (ADS)
May 2005	E	Non-Confidential	RVDS 2.2 Release
March 2006	F	Non-Confidential	RVDS 3.0 Release
March 2007	G	Non-Confidential	RVDS 3.1 Release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited. Other brands and names mentioned herein may be the trademarks of their respective owners.

CodeWarrior® and Metrowerks® are registered trademarks of Metrowerks, Inc.

PowerParts™ is a trademark of Metrowerks, Inc.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Contents
RealView Development Suite v3.1 CodeWarrior IDE Guide

Preface
About this book x
Feedback xv

Chapter 1 Introduction
1.1 About the CodeWarrior IDE 1-2
1.2 About the CodeWarrior IDE for RVDS 1-4

Chapter 2 Working with Files
2.1 About working with files 2-2
2.2 Creating and opening files 2-3
2.3 Saving files 2-12
2.4 Closing files 2-16
2.5 Printing files 2-18
2.6 Reverting to the most recently saved version of a file 2-20
2.7 Comparing and merging files and directories 2-21

Chapter 3 Working with Projects
3.1 About working with projects 3-2
3.2 Overview of the project window 3-4

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.3 Working with simple projects 3-11
3.4 Working with project windows 3-18
3.5 Working with project workspaces 3-22
3.6 Working with project stationery 3-24
3.7 Managing files in a project 3-25
3.8 Configuring the CodeWarrior IDE for complex or multi-user projects 3-38
3.9 Working with multiple build targets and sub-projects 3-40

Chapter 4 Working with the ARM Project Stationery
4.1 About the ARM Project stationery 4-2
4.2 Working with an existing ARM project 4-5
4.3 Using the Thumb ARM interworking stationery 4-7
4.4 Creating binary ROM images 4-13
4.5 Creating a project that builds an object library and ROM images 4-14
4.6 Creating a project with a sub-project 4-23
4.7 Creating your own project stationery 4-29
4.8 Converting ARM projects to Thumb projects 4-32
4.9 Converting Executable Image projects to Library projects 4-34
4.10 Compiling and linking a project 4-35
4.11 Processing output 4-44
4.12 Running batch files with the Batch File Runner 4-48

Chapter 5 Working with the RealView Debugger
5.1 About working with the RealView Debugger 5-2
5.2 Generating debug information 5-3
5.3 Running and debugging your code 5-7
5.4 Using the message window 5-9

Chapter 6 Editing Source Code
6.1 About editing source code 6-2
6.2 Overview of the editor window 6-3
6.3 Configuring the editor window 6-7
6.4 Editing text 6-10
6.5 Navigating text 6-19

Chapter 7 Searching and Replacing Text
7.1 About finding and replacing text 7-2
7.2 Finding and replacing text in a single file 7-3
7.3 Finding and replacing text in multiple files 7-8
7.4 Using grep-style regular expressions 7-17

Chapter 8 Working with the Browser
8.1 About working with the browser 8-2
8.2 Enabling browser data production 8-5
8.3 Using browser views 8-8
8.4 Using the browser 8-21

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.5 Creating classes and members with browser wizards 8-30

Chapter 9 Configuring IDE Options
9.1 About configuring the CodeWarrior IDE 9-2
9.2 Overview of the IDE Preferences window 9-3
9.3 Choosing general preferences 9-6
9.4 Choosing editor preferences 9-21
9.5 Setting commands and key bindings 9-37
9.6 Customizing toolbars 9-48

Chapter 10 Configuring a Build Target
10.1 About configuring a build target 10-2
10.2 Overview of the Target Settings window 10-4
10.3 Configuring general build target options 10-8
10.4 Using the Equivalent Command Line text box 10-35
10.5 Configuring assembler and compiler language settings 10-37
10.6 Configuring linker settings 10-65
10.7 Configuring editor settings 10-82
10.8 Configuring the debugger 10-84
10.9 Configuring miscellaneous features 10-86

Appendix A Running the CodeWarrior IDE from the Command Line and Command Window
A.1 Using the CodeWarrior IDE from the command line A-2
A.2 Using the CodeWarrior Command Window A-4

Appendix B CodeWarrior IDE Installation and Preference Settings
B.1 The CodeWarrior IDE preferences directory B-2
B.2 Using different versions of the CodeWarrior IDE B-3
B.3 Using the cwfileassoc tool B-4

Appendix C CodeWarrior Project Conversion Utilities
C.1 Converting old CodeWarrior projects C-2
C.2 Converting GNU Make files C-5
C.3 Using an external Make file C-6
C.4 Converting RealView Debugger project files C-14

Appendix D CodeWarrior IDE Reference
D.1 CodeWarrior IDE menu reference D-2
D.2 CodeWarrior IDE default key bindings D-17

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Preface

This preface introduces the CodeWarrior® *Integrated Development Environment* (IDE) and its documentation. It contains the following sections:

- *About this book* on page x
- *Feedback* on page xv.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

About this book

This book provides user information for the CodeWarrior IDE for *RealView® Development Suite* (RVDS). It describes the major graphical user interface components of the CodeWarrior IDE and provides information on features specific to developing software for ARM® architecture-based processors.

Intended audience

This book is written for all developers who are developing applications for ARM architecture-based processors, and who are using the ARM Limited version of the CodeWarrior IDE to manage those development projects under all current versions of Windows. It assumes that you are an experienced software developer and that you are familiar with the ARM RealView development tools, as described in the *RealView Development Suite Getting Started Guide*. It does not assume that you are familiar with the CodeWarrior IDE.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the CodeWarrior IDE and for a summary of how it is used with the RealView development tools.

Chapter 2 ***Working with Files***

Read this chapter for details of how to work with source files in the CodeWarrior IDE. This chapter provides basic information on managing your source files, and describes how to use the CodeWarrior IDE file comparison and merging functions.

Chapter 3 ***Working with Projects***

Read this chapter for details of how to use the CodeWarrior IDE project files to organize your project source files, and specify the output from compiling and linking your source. This chapter gives details of how to structure multiple build targets, control dependencies between build targets, and use other structural elements of CodeWarrior IDE projects.

Chapter 4 ***Working with the ARM Project Stationery***

Read this chapter for information about creating projects using the ARM project stationery provided with the CodeWarrior IDE, and how to use and modify the default stationery to generate ARM and Thumb® executable images, libraries, and disassembled code listings.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 5 ***Working with the RealView Debugger***

Read this chapter for details of how to use the RealView Debugger with the CodeWarrior IDE. It describes how the CodeWarrior IDE interacts with the RealView Debugger. It also describes parts of the CodeWarrior IDE that are useful for finding errors in your code, such as the message window.

Chapter 6 ***Editing Source Code***

Read this chapter for details of how to use the CodeWarrior IDE built-in text editor. It describes the basic functionality of the CodeWarrior IDE editor, and provides information on useful file navigation techniques that enable you to find related header and source files, find function definitions, and add markers to your source code.

Chapter 7 ***Searching and Replacing Text***

Read this chapter for details of how to use the CodeWarrior IDE find and replace facility to search text files and replace found text. It also describes the batch search facilities that enable you to search multiple source files and directories, and define named file lists for search operations.

Chapter 8 ***Working with the Browser***

Read this chapter for details of how to use the CodeWarrior IDE browser to view your source code from a number of object-oriented perspectives, including class-based and inheritance-based views.

Chapter 9 ***Configuring IDE Options***

Read this chapter for details of how to set CodeWarrior IDE configuration options that apply across all projects. It describes general interface options, editor options, and syntax coloring options. It also gives information on configuring command key binding and the CodeWarrior IDE toolbars.

Chapter 10 ***Configuring a Build Target***

Read this chapter for important information on configuring target-specific options for build targets within your projects. It describes how to use the CodeWarrior IDE to configure options for the RealView development tools, to produce machine code for execution on an ARM processor.

Appendix A ***Running the CodeWarrior IDE from the Command Line and Command Window***

Read this appendix for information on using the CodeWarrior IDE from the command line.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Appendix B ***CodeWarrior IDE Installation and Preference Settings***

Read this appendix for information on installing multiple copies of the CodeWarrior IDE and using the CodeWarrior IDE for RVDS with other versions of the CodeWarrior IDE. This appendix also describes the `cwfileassoc` tool used to modify CodeWarrior file associations.

Appendix C ***CodeWarrior Project Conversion Utilities***

Read this appendix for information on the project conversion utilities provided by CodeWarrior that can be used to import projects created by other IDE applications. This appendix also describes how to create custom projects to make use of an external build system.

Appendix D ***CodeWarrior IDE Reference***

Read this appendix for a summary of the CodeWarrior IDE menu commands and default key bindings.

This book assumes that the ARM software is installed in the default location. For example, on Windows this might be *`volume`*`:\Program Files\ARM`. This is assumed to be the location of *`install_directory`* when referring to path names. For example, *`install_directory`*`\Documentation\...`. You might have to change this if you have installed your ARM software in a different location.

Typographical conventions

The following typographical conventions are used in this book:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`monospace` Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

bold
Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM processor signal names.

...
In the middle of a path name ... indicates that additional directories exist between the directory names specified. The unspecified path names are usually version and build numbers, and platform-specific directory names, For example:

install_directory`\IDEs\CodeWarrior\CodeWarrior\...\...\...\Help`

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for the ARM family of processors.

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda, and the ARM Frequently Asked Questions (FAQs).

ARM publications

This book contains information that is specific to the latest version of the CodeWarrior IDE supplied with RVDS. See the following publications for information on the *RealView Compilation Tools* (RVCT) and the RealView Debugger:

- *RealView Development Suite Getting Started Guide (ARM DUI 0255)*
- *RVCT Essentials Guide* (ARM DUI 0202)
- *RVCT Compiler User Guide* (ARM DUI 0205)
- *RVCT Compiler Reference Guide* (ARM DUI 0348)
- *RVCT Libraries and Floating Point Support Guide* (ARM DUI 0349)
- *RVCT Linker and Utilities Guide* (ARM DUI 0206)
- *RVCT Assembler Guide* (ARM DUI 0204)
- *RVCT Developer Guide* (ARM DUI 0203)
- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Development Suite Glossary* (ARM DUI 0324).

Other publications

This book provides information specific to the ARM version of the Metrowerks CodeWarrior IDE. For more information on Metrowerks, and the CodeWarrior IDE generally, including version control plug-in availability, visit the CodeWarrior web site at `http://www.codewarrior.com`.

The following book is referenced in the text:

Friedl, J.E.F., *Mastering Regular Expressions, Powerful Techniques for Perl and Other Tools*, 1997. O'Reilly & Associates, Inc. ISBN 1-56592-257-3.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Feedback

ARM Limited welcomes feedback on both RealView Development Suite and the documentation.

Feedback on the RealView Development Suite

If you have any problems with RVDS, contact your supplier. To help them provide a rapid and useful response, give:

- your name and company
- the serial number of the product
- details of the release you are using
- details of the platform you are running on, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on this book

If you have any problems with this book, please send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 1
Introduction

This chapter introduces the CodeWarrior IDE. It contains the following sections:

- *About the CodeWarrior IDE* on page 1-2
- *About the CodeWarrior IDE for RVDS* on page 1-4.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

1.1 About the CodeWarrior IDE

The CodeWarrior IDE provides a simple, versatile, graphical user interface for managing your software development projects. You can use the CodeWarrior IDE for *RealView® Development Suite* (RVDS) to develop C, C++, and ARM assembly language code targeted at ARM processors. It speeds up your build cycle by providing:

- comprehensive project management capabilities
- code navigation routines to help you locate routines quickly.

The CodeWarrior IDE enables you to configure the ARM tools to compile, assemble, and link your project code.

Note

In this book, the terms *compile* and *compilation* apply generically to both compiling C/C++ source files and assembling ARM/Thumb® assembly language source files.

There are two distinct meanings of *target* in CodeWarrior IDE terminology:

Target system The specific ARM architecture-based hardware, or simulated hardware, for which you write code. For example, if you are writing code to run on an ARM development board, the development board is referred to as the target system.

Build target The collection of build settings and files that determines the output that is created when you build your project.

The CodeWarrior IDE enables you to organize source code files, library files, other files, and configuration settings into a *project*. Each project enables you to create and manage multiple configurations of build target settings. For example, you can compile a debugging build target and an optimized build target. Build targets can share files in the same project while using their own settings.

The CodeWarrior IDE includes:

- a source code editor that provides syntax coloring, code formatting, code completion, and is integrated with the CodeWarrior IDE browser
- a source code browser that keeps a database of symbols defined in your code, and enables you to navigate through your source code quickly and easily
- search and replace capabilities that enable you to use grep-style regular expressions, and perform batch searches through multiple files

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- file comparison capabilities that enable you to locate, and optionally merge the differences from one text file to another, and to compare the contents of directories.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

1.2 About the CodeWarrior IDE for RVDS

The CodeWarrior IDE for RVDS is based on Metrowerks CodeWarrior IDE version 5.7. A CodeWarrior plug-in has been developed by ARM to provide support for the RVDS toolchain. The plug-in provides:

- ARM-specific configuration panels that enable you to configure the ARM development tools from within the CodeWarrior IDE
- ARM-targeted project stationery that enables you to create basic ARM and Thumb projects from the CodeWarrior IDE.
- See Chapter 3 *Working with Projects* for more information.

  Note

  If you are setting up a complex project environment, see *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-38 for more information.

- Chapter 4 *Working with the ARM Project Stationery* for information on using the ARM project stationery provided with the CodeWarrior IDE for RVDS.

Although the plug-in has tightly integrated the RVDS toolchain with the CodeWarrior IDE, there are a number of areas of functionality that are not implemented by the ARM version of the CodeWarrior IDE. These are mostly related to debugging:

- There are a number of configuration panels that are not used by the RVDS toolchain, such as the Runtime Settings target configuration panel. See Chapter 10 *Configuring a Build Target* for more information.
- The RealView Debugger is not tightly integrated with the CodeWarrior IDE. For settings such as breakpoints or watchpoints you must use the debugger interface. See *How the RealView Debugger work with the CodeWarrior IDE* on page 5-2 for more information.
- There are a number of menu commands and windows that are not implemented by the ARM version of the CodeWarrior IDE. These are described in *Unused menu commands and windows* on page 1-5.

If you are familiar with the CodeWarrior IDE in other environments, you will also notice that some areas of functionality have been removed completely for the ARM version of the CodeWarrior IDE, such as Rapid Application Development templates. These templates are more suitable for creating desktop GUI applications and are not relevant to embedded software design.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

1.2.1 Unused menu commands and windows

The following commands and windows are not implemented in the ARM version of the CodeWarrior IDE:

File menu

The following items are not used by the CodeWarrior IDE for RVDS:

- Import Components...
- Close Catalog...

Edit menu

The following item is not used by the CodeWarrior IDE for RVDS:

- Design Settings...

View menu

The following items are not used by the CodeWarrior IDE for RVDS:

- Symbolics
- Breakpoints
- Registers
- Expressions
- Global Variables.

Project menu

The following items are not used by the CodeWarrior IDE for RVDS:

- Precompile.

Debug menu

Commands in the Debug menu do not apply to ARM. See Chapter 5 *Working with the RealView Debugger* for more information.

Data menu

Commands in the Data menu do not apply to ARM.

Browser menu

The following items are not used:

- New Property
- New Method
- New Event Set
- New Event.

Catalog menu

Commands in the Catalog menu do not apply to ARM.

Layout menu

Commands in the Layout menu do not apply to ARM.

For a detailed list of menu commands, see Appendix D *CodeWarrior IDE Reference*.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

1.2.2 Converting other IDE projects and files

CodeWarrior for RVDS can convert projects or files created using other development environments into CodeWarrior for RVDS projects:

- Projects created using earlier versions of the CodeWarrior IDE can be directly imported into CodeWarrior for RVDS.
- CodeWarrior for ADS projects (v1.0, v1.0.1, v1.1 and v1.2) can be directly imported into CodeWarrior for RVDS.
- CodeWarrior projects can be exported as GNU `Make` files. However, `Make` files cannot be directly imported into CodeWarrior for RVDS (see *Importing GNU Make files*).
- XML files (formatted in CodeWarrior for RVDS XML format) can be directly imported into CodeWarrior for RVDS.
- RealView Debugger project files must be saved to XML format before they can be imported into CodeWarrior for RVDS.
- Projects can be imported into the CodeWarrior IDE to use the project manager, and editor, but can still be built with your original `Make` files.

For more information about the conversion utilities, see Appendix C *CodeWarrior Project Conversion Utilities*.

Importing GNU `Make` files

If you are upgrading to RVDS v3.1 from release RVDS v2.2 or earlier, be aware that GNU `Make` files cannot be imported into CodeWarrior. The CodeWarrior IDE for RVDS v2.2 contained a GNU `Make` file Importer wizard. However, this feature is not supported in RVDS v3.1.

1.2.3 About the CodeWarrior example projects

This book references examples provided with RVDS in the main examples directory *`install_directory`*`\RVDS\Examples`. See Chapter 4 *Working with the ARM Project Stationery* for an example using the Dhrystone project, `dhrystone.mcp`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 2
Working with Files

This chapter describes how to work with source files in the CodeWarrior IDE. It contains the following sections:

- *About working with files* on page 2-2
- *Creating and opening files* on page 2-3
- *Saving files* on page 2-12
- *Closing files* on page 2-16
- *Printing files* on page 2-18
- *Reverting to the most recently saved version of a file* on page 2-20
- *Comparing and merging files and directories* on page 2-21.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.1 About working with files

This chapter describes how to use the CodeWarrior IDE to perform basic operations on files, including source files, project files, and text files. It describes basic file operations such as opening, closing, saving, and printing.

In addition, it describes how to use more sophisticated features of the CodeWarrior IDE, for example:

- navigating between related files (see *Switching between source and header files* on page 2-11)
- using the built-in file comparison features to compare and merge one or more files (see *Comparing and merging files and directories* on page 2-21).

This chapter does not provide detailed information on editing or managing files within a project. See:

- Chapter 3 *Working with Projects* for information on how source files fit into the project structure
- Chapter 6 *Editing Source Code* for information on how to use the CodeWarrior IDE editor to edit files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.2 Creating and opening files

There are several ways to open a file with the CodeWarrior IDE. This section describes:

- *Creating a new file*
- *Opening files from the File menu* on page 2-5
- *Opening files from the project window* on page 2-6
- *Opening header files from an editor window* on page 2-9.

Note

You cannot open libraries with the CodeWarrior IDE editor. However, you can add libraries to a project and link them into your project.

2.2.1 Creating a new file

To create a new text file:

1. Select **New…** from the **File** menu and click the **File** tab in the New dialog box. The CodeWarrior IDE displays the File panel with a list of new file types (see Figure 2-1 on page 2-4).

   Note

   The Component Catalog File and the PowerParts Palette Wizard entries, and the Object tab, are not used by the ARM® version of the CodeWarrior IDE. For more information about using the CodeWarrior IDE with PowerParts, see the Metrowerks web site.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-1 New dialog box

2. Click **Text File** to create a new Text file.

3. Enter a filename for the new file in the **File name** field. If you want to add the new file to a build target in an existing project, you must ensure that the filename you enter uses a filename extension that is defined in the File Mappings configuration panel for the build target. See *Configuring file mappings* on page 10-25 for more information.

4. Enter the path to the new file in the Location field, or click **Set…** and select the directory from the standard file dialog box.

5. Click **Add to Project** if you want to add the new file to an existing project, see Figure 2-2 on page 2-5:

   a. Click the **Project** drop-down menu to see a list of currently open projects. Select the project to which you want to add the file.

      The Targets field displays a list of the build targets defined for the project you select.

   b. Select the build targets to which you want to add the source file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-2 Add new file to project

6. Click **OK** to create the new file. The new file is added to the selected build targets.
7. Enter your text or source code in the new file, as required. See Chapter 6 *Editing Source Code* for more information on editing text.
8. Save the text file. See *Saving files* on page 2-12 for more information.

Note

If you do not associate the file with a project, you can do this later. See *Adding files to a project* on page 3-26 for more information.

2.2.2 Opening files from the File menu

You can open two types of file in the CodeWarrior IDE:

- Text files, such as a source code file or a header file.
- Project files. See *Working with simple projects* on page 3-11 for information on opening projects.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Opening text files

To open a text file or a source code file:

1. Select **Open…** from the **File** menu. The CodeWarrior IDE displays an Open dialog box (Figure 2-3).

Figure 2-3 Open dialog box

2. Select the file you want to open and click **Open**. The CodeWarrior IDE opens the file in an editor window. See Chapter 6 *Editing Source Code* for more information on editing the file you have opened.

2.2.3 Opening files from the project window

There are a number of ways to open files from the project window, depending on the type of file you want to see. These are:

Using the File column

Use this column to open a file that is in the project.

Using the Group drop-down menu

Use this drop-down menu to open a text source file from within a collapsed group.

Using the Header Files drop-down menu

Use this drop-down menu to open a header file included from a project source file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Opening files from the File column

To open a file from the File column:

1. Select the file or files you want to open from the **File** column in the **File view** or **Link Order** view of the project window. See *Selecting files and groups* on page 3-25 for information on selecting multiple files in a project.

2. Open the selected files. Either:
   - double-click the selected files
   - press the Enter key.

   The CodeWarrior IDE opens selected text files in an editor window. If project files are selected, the CodeWarrior IDE opens the project. If library files are selected, they are ignored.

See *Overview of the project window* on page 3-4 for more information on the File column.

Opening files from the Group drop-down menu

You can open a source file from the drop-down menu for the group that contains the file, even if the group is collapsed and the file is not visible in the project window.

To open a file from the Group drop-down menu:

1. Select the group that contains the source file you want to open.

2. Click the pop-up button for the group. A Group drop-down menu is displayed that contains a menu item for each file in the group. Figure 2-4 on page 2-8 shows an example.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-4 Group drop-down menu

3. Select the source file that you want to open. The file opens in an editor window.

See *Grouping files in a project* on page 3-30 for more information about creating groups of files in a project.

Opening header files from the Header Files drop-down menu

To open a header file that is associated with a source file:

1. Select the source file in the project window.
2. Click the **Header Files** drop-down menu. A list of header files is displayed, as shown in Figure 2-5 on page 2-9.

   Two types of header file are displayed:

   - Header files enclosed in angle brackets <...> are system header files found in the System access paths. See *Configuring access paths* on page 10-12 for information on how the CodeWarrior IDE searches for system header files.
   - Header files that are not enclosed in angle brackets are header files found in the User access paths. You might have created these header files yourself, or been supplied with them in order to use exported functions of a library. See *Configuring access paths* on page 10-12 for more information on how the CodeWarrior IDE searches for user header files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-5 Header Files drop-down menu in the project window

3. Select the header file you want to open from the list. The CodeWarrior IDE opens the header file in an editor window.

Note

- Press Ctrl+` to switch between a source file and its header file. See *Switching between source and header files* on page 2-11 for more information.
- If you click the **Header Files** drop-down menu for a library file that is part of your project, you only have the option to Touch or Untouch the library file. You cannot open the corresponding header file for a library file in this way. See *Touching and untouching files* on page 3-34 for more information on touching files.

2.2.4 Opening header files from an editor window

This section describes how to open header files from an editor window:

- *Open a header file using the Header Files drop-down menu*
- *Opening a header file with the Find and Open menu item* on page 2-10
- *Switching between source and header files* on page 2-11.

Open a header file using the Header Files drop-down menu

To open a header file from a source file you are editing:

1. Click the **Header Files** drop-down menu at the top left of the editor window.

   The drop-down menu lists all header files used by the source file.

2. Select a file from the list displayed in the **Header Files** drop-down menu to open it in a new editor window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

If there are no files available in the drop-down menu, it means one or both of the following:

- your text file does not contain any source code
- the source file does not include any header files.

Opening a header file with the Find and Open menu item

You can use the Find and Open menu item to open header files in two different ways:

- If you are editing a source file and the source file contains the name of a header file:
  1. Select the name of the header file you want to open. For example, if the source file contains:

     ```
     #include <stdio.h>
     #include <string.h>
     // source code
     ```

     you can double-click on `stdio` or `string` to select it. You do not need to select the `.h` part of the name.
  2. To open the selected header file, use one of the following:
     - select **Find and Open File...** from the **File** menu
     - right-click and select **Find and Open File...** from the menu
     - press Ctrl+D.

     The CodeWarrior IDE searches for the selected file and opens it in an editor window. If the file cannot be found, a system beep sounds. The CodeWarrior IDE uses the settings specified in the Access Paths configuration panel.
- If you are editing a source code file and want to open a file without selecting any text:
  1. Open the Find and Open File dialog box using one of the following:
     - select **Find and Open File...** from the **File** menu
     - right-click and select **Find and Open File...** from the menu
     - press Ctrl+D

     The CodeWarrior IDE displays the Find and Open File dialog box (see Figure 2-6 on page 2-11).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-6 Find and Open File dialog box

2. Type the name of the header file you want to open in the **Open** text field. The CodeWarrior IDE uses the settings specified in the Access Paths configuration panel to search for header files.
3. Select the **Search Only in System Paths** option to restrict the search to the directories specified in the System Paths pane of the Access Paths configuration panel.

   Do not select the **Search Only in System Paths** option to search both System Paths pane and User Paths pane (all paths specified in the Access Paths).

See *Configuring access paths* on page 10-12 for more information on configuring access paths.

Switching between source and header files

You can use the Ctrl+` keyboard shortcut to switch between a header file and its corresponding source file. To do this, your header file must have the same name as your source file, except for the file extension.

For example, if you are editing `myFile.cpp` and you want to see the associated header file, press Ctrl+` to display `myFile.h` in a new editor window. Type the same keyboard shortcut again to switch back to `myFile.cpp` file.

The CodeWarrior IDE searches the project directories defined in the Access Paths configuration panel to find the header file. See *Configuring access paths* on page 10-12 for more information on configuring access paths.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.3 Saving files

This section describes the ways that the CodeWarrior IDE can save files. It contains the following sections:

- *Saving project files*
- *Saving editor files*
- *Saving a backup copy of a file* on page 2-14.

2.3.1 Saving project files

Projects are opened for exclusive read/write access and are continually updated on the disk. Projects are saved when they are closed, when you exit the CodeWarrior IDE, or when you select **Save A Copy As…** from the **File** menu. You do not have to save projects explicitly.

2.3.2 Saving editor files

You can save editor files either explicitly, or automatically when your project is built.

Saving one file

To save your changes to a single text file:

1. Ensure that the text window you want to save is the active window.
2. Select **Save** from the **File** menu. If the file exists, the CodeWarrior IDE saves the file.
3. If the file is a new and untitled file, the CodeWarrior IDE displays the Save As dialog box. Enter a new name and location for your file. See *Renaming and saving a file* on page 2-13 for more information.

Note

The Save menu item is disabled if:

- The window is new and has no data.
- The contents of the active window have already been saved, and have not been modified since the last save. Modifying a file and then undoing the change marks the file as modified.
- The active window is the project window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Saving all files

To save your changes to all the files currently open, select **Save All** from the **File** menu (or press the keyboard shortcut Ctrl+Shift+S). The CodeWarrior IDE editor saves all the modified files to your disk.

Saving files automatically

The CodeWarrior IDE can automatically save changes to all your modified files whenever you select any of the following options from the Project menu:

- Preprocess
- Compile
- Disassemble
- Bring Up To Date
- Make
- Debug
- Run.

You can use the Save open files before build IDE preferences option to save your work before building and running your program. If you are experimenting with a change and do not want to save changes, turn this option off.

Caution

RealView Debugger read source files from disk. If you are debugging at source level and select this option, the debugger does not read any unsaved modifications to the source.

See the description of the Save open files before build IDE preferences option in *Configuring build settings* on page 9-6 for more information.

Renaming and saving a file

To save a text file under a new name:

1. Ensure that the text window you want to rename is the active window.
2. Select **Save As...** from the **File** menu. The CodeWarrior IDE displays the Save As dialog box (see Figure 2-7 on page 2-14).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-7 Save As dialog box

3. Enter the new name and location of the file.
4. Click **Save** to save the file under its new name. The CodeWarrior IDE saves the file and changes the name of the editor window to the name you entered. If the file is in the current project, the CodeWarrior IDE updates the project to use the new name.

Note

See *Saving a backup copy of a file* if you want to save a copy of a file, but you do not want to change the name of the file used in the project.

Saving as a Mac OS, or UNIX text file

The ARM-supplied version of the CodeWarrior IDE is supported on Windows platforms only. However, you can use the CodeWarrior IDE to open text files created on other platforms without changing the original format of the file.

See *Specifying Other Settings* on page 9-27 for more information on saving text files in a different text format.

2.3.3 Saving a backup copy of a file

You can save backup copies of both text and project files.

Saving a copy of a text file

To save a backup copy of a text file before you change the original:

1. Ensure that the text window you want to save is the active window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select **Save A Copy As…** from the **File** menu. The CodeWarrior IDE displays the Save A Copy As dialog box (Figure 2-8).

Figure 2-8 Save A Copy As dialog box

3. Type the name and location for the new file in the Object name text field.

4. Click **Save**. The CodeWarrior IDE saves a copy of the file with the new name. It does not change the file in the editor window or in the current project, and it does not change the currently-open project to use the new file.

Saving a copy of the current project

To save a copy of the current project:

1. Ensure that the project window you want to save is the active window.

2. Select **Save A Copy As…** from the **File** menu. The CodeWarrior IDE displays the Save a copy of project as dialog box.

3. Type the name you want to use for the copy of the project if you want to change the default.

4. Click **Save** to save the project.

See *Comparing XML-formatted projects* on page 2-28 for information on exporting a project to XML.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.4 Closing files

Every editor or project window in the CodeWarrior IDE is associated with a file. When you close the window, you close the file. This section describes:

- *Closing project files*
- *Closing editor files*.

2.4.1 Closing project files

To close a project file, select the project by clicking in the window (or on the tab if the project window is docked) and use any of the following methods to close the project:

- select **Close** from the **File** menu. (If the project window is docked, right-click over the tab and select **Close** from the menu.)
- click the **Close** button (or the smaller close button if the project window is docked).
- press Ctrl+F4 or double-click over the CodeWarrior icon in the top-left corner of the window.

Source files opened from the project remain open when you close the project. Projects are saved when you close the project window. See *Saving a project* on page 3-15 for more information on saving project files.

2.4.2 Closing editor files

This section describes how to close editor files.

Closing one file

To close an editor window, select the window by clicking in the window (or on the tab if the window is docked):

1. Use any of the methods described in *Closing project files* to close the editor window.

   If you have unsaved changes to the text file, the CodeWarrior IDE asks if you want to save the changes before closing the window (see Figure 2-9 on page 2-17).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-9 Unsaved changes alert

2. Click one of:
   - **Save**, if you want to save your changes before closing the file.
   - **Don't Save**, if you want to close the file without saving your changes. All unsaved changes are discarded.
   - **Cancel**, if you want to cancel the close and return to the editor window without saving your changes.

Note

The Close command also saves other properties of the window, such as the size, location, and the selected text in the active window. See *Editor settings* on page 9-24 for information on how to configure these options. If the appropriate options are enabled, the editor window occupies the same position on your screen and has the same text selected the next time you open the source code file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.5 Printing files

Use the print options in the CodeWarrior IDE to print open files, a project file, or the contents of a window.

The topics in this section are:

- *Setting print options*
- *Printing a window*.

2.5.1 Setting print options

To configure printing options:

1. Select **Print…** from the **File** menu. The CodeWarrior IDE displays the Print dialog box for your printer.
2. Use the dialog box to select the paper size, orientation, and other settings. The specific settings and options depend on the printer your computer is configured to use. See you printer and operating system documentation for more information on printer options.
3. Click **OK** to save your selected printer options.

2.5.2 Printing a window

To print a window:

1. Ensure that the window you want to print is the active window. If the active window is:
   - an editor window, the CodeWarrior IDE prints the text file associated with that window
   - a project window, the CodeWarrior IDE prints a screen representation of the project window.
2. Select **Print...** from the **File** menu. The Print dialog box for your printer is displayed.
3. Select your print options. The options available will vary depending on your printer. See your printer and operating system documentation for more information on print options.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

In addition, the Print dialog box contains CodeWarrior IDE-specific print options:

Selection Only

This option is only available if text is selected in an editor window. Select this option to print only the selected text. If this option is not selected, the CodeWarrior IDE prints the entire file.

Use Syntax Highlighting

Select this option to print the editor window with syntax coloring. On a black and white printer, colors are printed as shades of gray. If this option is not selected, the CodeWarrior IDE prints the file in black and white without syntax coloring.

Wrap Text Lines

Select this option to ensure that text lines in the window that are too long to fit across the width of the printed page are wrapped and printed on extra lines.

Print Line Numbers

Select this option to print line numbers alongside each line of the file.

4. Click **Print** in the Print dialog box. The CodeWarrior IDE spools the file to your printing software for printing.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.6 Reverting to the most recently saved version of a file

You can revert to the most recently saved version of a file if you have edited the file and you do not want to keep the changes you have made. To revert to the saved version of a file:

1. Ensure that the active window contains the changes that you want to discard.
2. Select **Revert** from the **File** menu. The CodeWarrior IDE displays a confirmation alert (Figure 2-10).

Figure 2-10 Revert to a previous file

3. Click **OK** to discard changes to the current file and open the last saved version of the file. All changes you have made since the last time you saved the file are discarded.

   Click **Cancel** if you do not want to revert to the last saved version of the file. The file you are working on is not changed or saved to disk, and you can continue editing it.

Note

- You can use multiple undo to retrace your changes in an editor file. See *Specifying Other Settings* on page 9-27 for more information.
- You can use the CodeWarrior IDE in conjunction with a *Version Control System* (VCS) to recover previous checked-in versions of your editor files. See the Metrowerks web site for information about the available VCS plug-ins for the CodeWarrior IDE.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2.7 Comparing and merging files and directories

You can use the CodeWarrior IDE to compare two text files, mark the differences between the files, and apply changes to the files. In addition, you can compare the contents of two directories.

The following sections show you how to use the CodeWarrior IDE file comparison features:

- *File comparison and merge overview*
- *Choosing files to compare* on page 2-23
- *Applying and unapplying differences* on page 2-24
- *Choosing directories to compare* on page 2-26
- *Comparing XML-formatted projects* on page 2-28.

2.7.1 File comparison and merge overview

Select **Compare Files** from the **Search** menu. The CodeWarrior IDE displays the Compare Files Setup dialog box that prompts you for a source file and a destination file to compare (see Figure 2-12 on page 2-24).

The file comparison window displays two text files, and the differences between them (see Figure 2-11 on page 2-22).

Differences are listed as insertions, deletions, and non-matching lines. The file comparison window has controls that enable you to examine, apply, and unapply the differences between the files. The currently selected difference is shown with a darker color and is outlined in black to contrast it from the other differences visible in the window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-11 File Compare Results window

The main parts of the File Compare Results window are:

Source file column

This column (on the left side of the File Compare Results window) displays the source text file that the CodeWarrior IDE uses as a basis for comparison with the destination file. You can edit this text.

Destination file column

This column (on the right side of the File Compare Results window) displays the destination file that is compared with the source file. Differences between the source file and the destination file can be added to, or removed from, the destination file. You can edit this text.

Comparison column

This column (between the source and destination columns) displays a graphical representation of where text was added or removed between the source and destination files.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Differences list

The Differences list (below the source and destination columns) details the insertions, deletions, and lines of mismatching text between the two files. Select an item in the list to display the difference in the source and destination columns. Text in the Differences list is displayed in italics when a difference is applied to the destination file.

Toolbar

The toolbar buttons (above the Differences list) enable you to apply or remove changes between the two files to the destination file. The toolbar buttons also enable you to undo and redo changes to the source and destination files.

Table 2-1 shows the control icons.

See *Customizing toolbars* on page 9-48 for information on how to customize the toolbar.

Table 2-1 File Compare Toolbar control icons

Control	Action
	Apply difference
	Unapply difference
	Undo
	Redo

2.7.2 Choosing files to compare

To open a file comparison window and select text files to compare:

1. Select **Compare Files...** from the **Search** menu. The CodeWarrior IDE displays the Compare Files Setup dialog box that prompts you for a source file and a destination file to compare (Figure 2-12 on page 2-24).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-12 Compare Files Setup dialog box

2. Click **Choose...** for each of the source and destination sections of the window to browse for the files to compare.

   Alternatively, you can also:

   - drag and drop files from the Windows desktop to the source and destination sections of the window
   - click on the Up/Down arrow button next to the **Choose...** button. A menu is displayed showing all the files that are currently open. Select a file to compare from the menu.

3. Set the Text Compare Options:

   Case sensitive

   Select this option to consider the case of characters as part of the comparison operation. To ignore case, deselect this option.

   Ignore extra space

   Select this option to ignore extra space and Tab characters at the beginning and end of lines.

   See *Choosing directories to compare* on page 2-26 for information on the **Folder Comparison Options**.

4. Click **Compare** to compare the two files and display the File Compare Results window.

2.7.3 Applying and unapplying differences

Use the toolbar and Differences list in the File Compare Results window to select the differences that you want to apply from the source file to the destination file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Applying a difference

To view and apply a difference from the source file to the destination file:

1. Click the entry for the difference in the Differences list.
2. Click the **Apply Difference** button in the toolbar, or select **Apply Difference** from the **Search** menu. The CodeWarrior IDE changes the destination file to match the source file for the selected difference. The applied difference is displayed in italics in the Differences list (Figure 2-13).

Figure 2-13 Applied difference

Applying all differences

To view and apply all reported differences between the source file and the destination file:

1. Click in the Differences list.
2. Select **Select All** from the **Edit** menu (or press Ctrl+A), to select all the reported differences.
3. Click the **Apply Difference** button in the toolbar, or select **Apply Difference** from the **Search** menu. All the reported differences are applied to the destination file. The applied differences are displayed in italics in the Differences list (Figure 2-14).

Figure 2-14 Applied all differences

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Unapplying a difference

To reverse a difference you have applied:

1. Click the entry for the difference you want to unapply. Applied differences are displayed in italics in the Differences list.
2. Click the **Unapply** button in the toolbar, or select **Unapply Difference** from the **Search** menu.

Note

The Apply Difference and Unapply Difference commands erase all actions from the Undo stack. When you exit the File Compare Results window after applying or unapplying differences, all undo and redo actions are cleared from the Undo stack.

2.7.4 Choosing directories to compare

You can use the CodeWarrior IDE comparison features to compare directories of files:

1. Select **Compare Files...** from the **Search** menu. The CodeWarrior IDE displays the Compare Files Setup dialog box, see Figure 2-15.

Figure 2-15 Compare Files Setup dialog box

2. Click **Choose…** for each of the source and destination compare sections.
3. Set the **Folder Compare Options**:

   Only show different files

   Select this option to display only files that are different in both directories in the **Files In Both Folders** list of the **Folder Compare Results** window, see Figure 2-16 on page 2-27. By default, this option is disabled, so all files in the source and destination directories are displayed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Comparisons between files in the source and destination directories are normally based on the file modification dates and file sizes. This is usually good enough to determine if there are differences between the two files. If there are invisible items in the directories, the comparison skips over those items.

Compare text file contents

Select this option to perform a more accurate comparison of the files in the two directories. The comparison performs a Compare Files command on every file in the source and destination directories and checks neither the modification dates nor the file sizes. The file comparison is slower, but the comparison information is more accurate.

Set the Text Compare Options. See *Choosing files to compare* on page 2-23 for information on the file comparison options.

4. Click **Compare** to compare the two directories. The CodeWarrior IDE displays the **Folder Compare** Results window, see Figure 2-16. The names of source code files, header files, text files, and directories are displayed in plain face. All other filenames are displayed in italics.

Figure 2-16 Folder Compare Results window

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The files are displayed in three lists:

Files in Both Folders

This list displays all files in both the source and destination directories, unless the **Only Show Different Files** option is enabled. Files that are different in the two directories have a small bullet to the right of their name.

Files Only in Source

This list displays all the files that exist only in the source directory.

Files Only in Destination

This list displays all the files that exist only in the destination directory.

5. Click on a filename in any of the lists to display specific information about the selected file in the Selected Item box at the bottom of the **Folder Compare Results** window.

6. Double-click on a file in the **Files In Both Folders** list to open a **Compare Files** window for resolving the differences between the two files.

Note

You can click on a zoom box ( ) for any of the three lists to expand them to fill the window. Click the zoom box again to collapse the lists back to their original size.

2.7.5 Comparing XML-formatted projects

The CodeWarrior IDE can export a project file to *eXtensible Markup Language* (XML) format. You can use the file comparison facility to compare two XML files, and merge the contents of the files. In addition, you can import a merged XML file into the CodeWarrior IDE and save the imported file as a new CodeWarrior IDE project.

To compare two CodeWarrior IDE projects and apply changes from one to the other:

1. Convert the required projects to XML format:
   a. Ensure that the same build target is currently selected for both project files when you export them to XML format. Otherwise, the Differences list might not reflect the differences in the two files accurately.
   b. Ensure that the project window for the first project is the currently active window.
   c. Select **Export Project...** from the **File** menu. The CodeWarrior IDE displays an Export project as dialog box (Figure 2-17 on page 2-29).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 2-17 Export project as dialog box

d. Save the project as an XML file.

e. Repeat the export for the second project.

2. Select **Compare Files...** from the **Search** menu. The CodeWarrior IDE displays the Compare File Setup dialog box (see Figure 2-12 on page 2-24).

3. Specify the XML-formatted versions of the two project files in the Compare Files Setup dialog box (see *Choosing files to compare* on page 2-23 for information on how to do this).

4. Click **Compare**. The CodeWarrior IDE displays the Differences list for the two projects. See Figure 2-11 on page 2-22 for an example of the File Comparison window.

5. Apply the changes you want to the destination file. See *Applying and unapplying differences* on page 2-24 for more information.

6. Select **Save** from the **File** menu to save your modified XML file.

7. Select **Import Project…** from the **File** menu. The CodeWarrior IDE displays a standard open file dialog box.

8. Select the XML file you want to import and click **Open**. The CodeWarrior IDE displays a Save As dialog box asking you to name the project file that will be created, and specify a location to save the project.

9. Enter the name and location of the new project, and click **Save**. The CodeWarrior IDE converts the XML-formatted file into a project file and saves the project file.

For more information on the format of CodeWarrior IDE XML projects and how to import XML projects into the CodeWarrior IDE see *Converting RealView Debugger project files* on page C-14.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 3
Working with Projects

This chapter introduces the CodeWarrior IDE project file and shows how to create, configure, and work with projects. It contains the following sections:

- *About working with projects* on page 3-2
- *Overview of the project window* on page 3-4
- *Working with simple projects* on page 3-11
- *Working with project windows* on page 3-18
- *Working with project workspaces* on page 3-22
- *Working with project stationery* on page 3-24
- *Managing files in a project* on page 3-25.
- *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-38
- *Working with multiple build targets and sub-projects* on page 3-40.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.1 About working with projects

CodeWarrior IDE projects are the highest level structural element that you can use to organize your source files and determine their output. This chapter describes many of the basic tasks involving projects, such as:

- creating projects
- opening projects
- adding files to projects
- saving projects
- moving files in the project window.

See Chapter 4 *Working with the ARM Project Stationery* for more information about ARM® project stationery and how to perform more complex project operations, such as:

- using the ARM-specific project stationery provided with the CodeWarrior IDE
- how to configure and use your own project stationery.
- how the CodeWarrior IDE uses project stationery
- creating nested projects
- creating multiple build targets
- dividing the project window into groups of files.

3.1.1 Project structure overview

A CodeWarrior IDE project is a collection of source files, library files, and other input files. You can organize the files in a project in various ways to provide a logical structure to your source. The most important structural element in a project is the *build target*. The build target defines how the source files within a project are processed, not the CodeWarrior IDE project itself.

Build targets

Every CodeWarrior IDE project defines at least one build target. A build target is a specific configuration of build options applied to all, or some of the source files in a project to produce an output file, such as an executable image, library, or code listing.

Complex projects can define up to 255 build targets. You can use multiple build targets to build different kinds of output files from one project file. For example, the ARM-supplied stationery projects define at least two build targets.

When you select build options for your project you apply them specifically to one or more build targets. See Chapter 10 *Configuring a Build Target* for detailed information on setting build options.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can define a specific build order for the build targets in a project, so that the CodeWarrior IDE builds one build target before building another, and optionally links the output from the build targets. This means that you can create a build target that depends on the output from some other build target. See *Assigning build target dependencies* on page 3-48 for more information.

Each build target in a project contains a collection of elements that the CodeWarrior IDE uses to build the output file. Build targets within a project can share some, or all, of their elements. A build target can include:

Source files and libraries

These are the basic input files for your project. They can be organized into groups, or included from other build targets and project files. You can use the project window to customize how individual source files are treated in a build target. For example, you can turn debugging on and off, compile, disassemble, preprocess, and check the syntax of individual source files.

You can also specifically exclude an individual source file from a build target. This enables you, for example, to have a proven but slow C language implementation of an algorithm for debugging, and an optimized assembly language implementation for product release.

Groups

These are groups of files and libraries. You can group related source files or libraries together to help organize your project sources conveniently.

Other build targets

You can use the output from one build target as input to another, or create independent build targets that generate different types of output. You can use dependent build targets, for example, to combine output objects from ARM and Thumb® build targets into a single output image. See *Working with multiple build targets and sub-projects* on page 3-40 for detailed information on build targets.

sub-projects

These are independent projects that you include in your main project. They can contain the same kinds of elements, such as files, build targets and additional sub-projects, as the main project. See *Creating sub-projects within projects* on page 3-53 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.2 Overview of the project window

The project window is the main CodeWarrior window from where you manage CodeWarrior IDE projects It shows information about the file and build targets in your project file (Figure 3-1).

Figure 3-1 Typical project window

The project window uses distinct views to display your files and build targets. These are described in detail in *Project views*.

3.2.1 Project views

The project window provides three distinct views on the files, groups, and sub-projects that make up your project (Figure 3-2).

Figure 3-2 Project window, view tabs

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To choose a view, click the tab at the top of the project window, as shown in Figure 3-2 on page 3-4.

The project views are described in:

- *Files view*
- *Link Order view* on page 3-8
- *Targets view* on page 3-10.

Files view

The Files view shows a list of all the files, groups, and sub-projects for all the build targets in the project (see Figure 3-3). Use this view to arrange your project into hierarchical groups without affecting the way the CodeWarrior IDE handles a build target. This view also displays other information, including modification status, file access paths, code size, data size, current build target, and debugging status.

Figure 3-3 Project window, Files view

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The Files view contains the following columns:

File column

The File column lists project files and groups in a configurable hierarchical view. A group can contain files and other groups. You can:

- Double-click a source filename in the File column to open the file in the CodeWarrior IDE editor, or a third-party editor set in the IDE Preferences window.
- Click the hierarchical control to display and hide the contents of groups.
- Right-click on a filename to display a context menu of commands that can be applied to the file. For example, to display the location of the file, right-click the filename in the project window and select **Open in Windows Explorer** from the context menu.

The File column in the Files view displays all files in the current project, whether or not they are included in the current build target.

Code column

The Code column shows the size, in bytes or kilobytes, of the compiled executable object code for files. For a group, the value is the sum of the sizes for files in the group in the current target. If `0` is displayed in the Code column, your file has not yet been compiled. If `n/a` is displayed, the file is not included in the current build target.

The values in this column do not necessarily reflect the amount of object code that will be included in the final output file. By default, the linker removes unused sections from input object files. See the description of unused section elimination in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

Data column

The Data column shows the size, in bytes, kilobytes (K), or megabytes (M). This includes zero-initialized data, but not stack space used by the object code for files in the project. If `0` is displayed in the Data column, the file has not yet been compiled, or no data sections are generated from this source code. If `n/a` is displayed, the file is not included in the current build target.

Like the Code column sizes, the data values listed in the Data column are not necessarily all added to the final output file. You can use the `fromelf` utility to determine the sizes of code and data sections in the final output image. See the description of `fromelf` in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Debug column

The Debug column indicates whether debugging information will be generated for individual files in a project if the RealView® compiler and assembler are not configured to generate debug information for all files in the build target.

A black marker in this column next to a filename or group name indicates that debugging information will be generated for the corresponding item. A gray marker next to a group name indicates that debugging information will be generated for only some of the files in the group.

To generate debugging information for a:

- file, click in the Debug column next to the file
- group, click in the Debug column next to the group
- project, Alt+click in the Debug column
- target, modify the assembler and compiler preferences.

See Chapter 5 *Working with the RealView Debugger* for detailed information on how debug information is generated for files.

Target column

The Target column indicates whether an item is in the currently selected build target. The CodeWarrior IDE displays this column if a project has more than one build target. A dark marker in this column next to a file or group means that the corresponding item is in the current build target. A gray marker next to a group indicates that only some of the files in that group are in the current build target.

To assign or unassign a current build target for a:

- file, click in the Target column next to the file
- group, click in the Target column next to the group
- project, Alt+click in the Target column.

See *Assigning files to build targets* on page 3-45 for information on adding or removing a file to or from a build target using the Target column.

Touch column

The Touch column indicates whether a file is marked to be compiled, assembled, or imported (libraries and object files). A marker in this column next to a filename or group name indicates that the corresponding item will be rebuilt at the next **Bring Up To Date**, **Make**, **Run** or **Debug** command. A gray marker next to a group indicates that only some of the files in that group are marked for compilation or assembly.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To touch or untouch:

- a file, click in the Touch column next to the file
- a group, click in the Touch column next to the group
- a project, Alt+click in the Touch column.

See *Synchronizing modification dates* on page 3-35 for more information.

Header Files drop-down menu

The **Header Files** drop-down menu:

- lists header files for your project source files
- enables you to open the selected item
- enables you to touch or untouch the selected item and set other options.

For groups, the **Header Files** drop-down menu lists the files within the group. See *Opening header files from the Header Files drop-down menu* on page 2-8 for more information.

File Control context menu

The **File Control** context menu is displayed by right-clicking on a filename or group name in the project window. The items displayed on the menu differ depending on whether a filename or group name is selected.

Link Order view

The Link Order view shows information about how the CodeWarrior IDE compiles and links the final output file for the current build target. Figure 3-4 on page 3-9 shows an example.

When you add files to a project, they are added in the same order in both the Files View and the Link Order view. This means that, by default, the CodeWarrior IDE compiles project files in the order shown in the Files view. You can change the order files are compiled by rearranging them in the Link Order view. In addition, files are displayed in the Link Order view only if they are included in the current build target.

Changing the order of files in the Link Order view can change the order in which the object code is placed in the final binary output produced from your project. The CodeWarrior IDE invokes the ARM linker with a list of object files in the order in which they are compiled. By default, the linker processes object files in presentation order.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

The Link Order view is a convenient way to control the order in which source files are processed. However, in general it is not advisable to depend on the Link Order view to control output image structure. The RealView Linker configuration panel provides limited control over section placement. For finer control, use a scatter-loading description file. See *Configuring the RealView Linker* on page 10-66 and the *RealView Compilation Tools Linker and Utilities Guide* for more information.

Figure 3-4 Link Order view

The Link Order view displays columns similar to those displayed in the Files view. The most important difference is that there is no Target column in the Link Order view. The Link Order view gives information only for those files that are included in the current build target. This means that a file *must* be selected in the Target column of the Files view in order to be displayed in the Link Order view.

The Link Order view contains the following columns:

File column

The File column lists the files in the current build target. Unlike the Files view, the Link Order view displays only files, not groups. Files are displayed in the order in which they will be compiled, regardless of whether they are in a group or not.

Code column, Data column, Debug column

These columns display the same information as in the Files view. See *Files view* on page 3-5 for more information.

Touch column

The Touch column indicates whether a file is marked to be compiled. A marker in this column next to a filename indicates that the corresponding item will be recompiled at the next **Bring Up To Date**, **Make**, **Run** or **Debug** command.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To touch or untouch:

- a file, click in the Touch column next to the file
- the current build target, Alt+click in the Touch column.

See *Synchronizing modification dates* on page 3-35 for more information.

Header Files pop-up menu

The **Header Files** pop-up menu:

- lists and opens header files for your source files
- enables you to touch or untouch the selected item and set other options.

See *Opening header files from the Header Files drop-down menu* on page 2-8 for more information.

File Control context menu

Right-click on an entry in the project window to display the **File Control** context menu. The **File Control** context menu provides context-specific commands, depending on the selected item.

Targets view

The Targets view shows information about the build targets in a project, and build target dependencies. Figure 3-5 shows an example Targets view with four targets defined by the ARM stationery.

Figure 3-5 Targets view

The Targets view shows a list of the build targets in the project. It also shows the objects, or sub-targets, that the build targets depend on to create a final output file.

See *Working with multiple build targets and sub-projects* on page 3-40 for more information.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.3 Working with simple projects

This section describes basic project operations:

- *Creating a new project*
- *Opening a project* on page 3-13
- *Closing a project* on page 3-15
- *Saving a project* on page 3-15
- *Choosing a default project* on page 3-16
- *Moving a project* on page 3-17
- *Importing and exporting a project as XML* on page 3-17.

3.3.1 Creating a new project

The CodeWarrior IDE can base new projects on an existing, preconfigured project stationery file that is used as a template for your new project. The CodeWarrior IDE provides two options for creating new projects:

Projects based on project stationery

Project stationery can be preconfigured with libraries and source code placeholders. Configuration options and build targets are predefined. However, ARM recommends that you review configuration options to ensure that they are relevant to your development environment. This kind of project file is useful for creating new projects quickly.

Empty projects

Empty projects do not contain any placeholder files or libraries, and use default values for all tool configuration options. If you choose to create a new empty project you must review all configuration options, define your own build targets, and add all required libraries and files.

See Chapter 10 *Configuring a Build Target* for information on configuring project, compiler, linker, and other target settings for your project. See *Working with project stationery* on page 3-24 for more information on project stationery.

To create a new project:

1. Select **New…** from the **File** menu. The CodeWarrior IDE displays a New dialog box (see Figure 3-6 on page 3-12).

   Note

   The New dialog box also contains a **File** tab and an **Object** tab. See *Creating a new file* on page 2-3 for more information on creating new source files. The **Object** tab is not implemented in the ARM version of the CodeWarrior IDE.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-6 New dialog box

2. Ensure that the **Project** tab is selected and specify the project stationery file on which you want to base your own project. You can select either:
   - An empty project, to create a project that contains no libraries or other support files.
   - An ARM or Thumb Executable image, library, or interworking project. See *Working with project stationery* on page 3-24 for more information on the project stationery files supplied by ARM. This is the recommended method if you are a new user.

   Note

   PowerParts wizards are not implemented in the ARM version of the CodeWarrior IDE. For more information about using the CodeWarrior IDE with PowerParts, see the Metrowerks web site.

   See *Using an external Make file* on page C-6 for information about the External Build wizard.

3. Enter a name for your project and either enter the project location or click **Set…** to specify a directory in which to store your project. By default, the CodeWarrior IDE adds a `.mcp` filename extension to the project filename.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Click **OK**. The CodeWarrior IDE creates a new project based on the project stationery you selected. See:
   - *Adding files to a project* on page 3-26, for information
   - *Compiling and linking a project* on page 4-35 for more information.

3.3.2 Opening a project

This section describes how to open:

- existing projects so you can work on them
- sub-projects from within a project window
- projects created on other platforms.

You can have more than one project open at a time. To switch to one of several open projects, select the project name from the **Window** menu. See *Choosing a default project* on page 3-16 for information on how to select one of your open projects as the default target for project-level commands.

To open a project file:

1. Select **Open** from the **File** menu. The CodeWarrior IDE displays an Open dialog box (see Figure 3-7).

Figure 3-7 Open dialog box

2. If not already set, use the **Objects of** drop-down menu to select **Project Files**. The file list changes to show the project files that you can open.

3. Select the project file you want to open and click **Open**. The CodeWarrior IDE opens the project and displays it in a project window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

If the project was created with an older version of the CodeWarrior IDE, you will be prompted to convert the older project to the newest version. If you decide to update, the CodeWarrior IDE saves a backup of the project and then converts the project to the newest version.

Using the Open Recent command

The CodeWarrior IDE maintains a list of the projects, files, and workspaces opened recently in the **File** menu. Use the **Open Recent** command to reopen one of these projects. See *Configuring IDE extras* on page 9-9 for information on setting the number of files that the CodeWarrior IDE stores in this menu.

Opening sub-projects from the project window

To open a sub-project contained within your project, double-click the sub-project file icon in the project window. The CodeWarrior IDE displays the sub-project in a new project window. See *Working with multiple build targets and sub-projects* on page 3-40 for more information.

Opening project files created on other host platforms

Project files are compatible across different platforms. For example, you can open and use a project created under Mac OS on Windows.

Note

The ARM version of the CodeWarrior IDE is supported on Windows only. However, this feature might be useful if you are moving to ARM from another target environment that also uses CodeWarrior development tools.

To use a project created on another host platform:

1. Ensure that the project has a `.mcp` filename extension. The CodeWarrior IDE uses this filename extension to recognize project files. If the three-letter extension is not present, the CodeWarrior IDE is unable to identify the project file.
2. Copy only the project file, not its associated `Data` directory, from the other host platform to your computer.
3. Open the project in the CodeWarrior IDE and rebuild it.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.3.3 Closing a project

To close a project:

1. Ensure that its project window is the currently active window.
2. Select **Close** from the **File** menu, or click the Windows **Close** button.

You do not have to close your project before quitting the CodeWarrior IDE application, because your project settings are automatically saved. See *Saving a project* for details of how the CodeWarrior IDE saves project information.

The CodeWarrior IDE enables you to have more than one project open at a time, so you do not have to close a project before switching to another project.

Note

Having multiple projects open at a time uses more memory, and also causes project opening times to lengthen slightly.

3.3.4 Saving a project

The CodeWarrior IDE automatically updates and saves your project when you perform certain actions. This section describes the actions that cause a project file to be saved.

Your settings are saved when you:

- close the project
- change the Preferences or Target Settings for the project
- add or delete files for the project
- compile any file in the project
- edit groups in the project
- remove object code from the project
- exit the CodeWarrior IDE.

You do not have to save your project manually unless you want to create a copy of it.

Information saved with your project

When the CodeWarrior IDE saves your project, it saves the following information:

- the names of the files added to your project and their locations
- all configuration options
- dependency information, such as the touch state and header file lists
- browser information
- references to the object code of any compiled source code files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

You can also save project window layout settings to a separate workspace file. For more information, see *Working with project workspaces* on page 3-22.

Saving a copy of your project

If you want to save a backup copy of a project file before you make some changes to the original, select **Save a Copy As…** from the **File** menu. The CodeWarrior IDE creates a copy of the project file under a new name that you specify, and leaves the original project file unchanged. The CodeWarrior IDE does not change the currently open project to use the new filename.

Caution

Do not attempt to make a copy of an open project from the Windows desktop. This might cause the project file to be corrupted. Always close the project before copying the project file.

3.3.5 Choosing a default project

The CodeWarrior IDE enables you to have more than one project open at a time. When you select some project-level commands, such as **Debug** or **Bring up to date** the CodeWarrior IDE applies the command in the following way if there is more than one project open:

- if one of the project windows is the currently active window, the CodeWarrior IDE applies the command to that project.
- if no project window is the currently active window and it is ambiguous as to which project the command should be applied to, the CodeWarrior IDE applies the command to the default project.

To specify a default project, select **Project** → **Set Default Project** → ***Project_name*** where *Project_name* is the name of the open project you want to make the default.

When you start the CodeWarrior IDE, the first project you open becomes the default project. If you close the default project with other projects still open, the default project becomes the project with the front-most project window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.3.6 Moving a project

The CodeWarrior IDE stores all the information it requires about a project in the project file. The project data directory contains additional information such as window positions, debug info, browser data, and other settings. However, the CodeWarrior IDE does not need these files to recreate your project.

To move a project, drag the project file (ending in `.mcp` if it obeys the project file naming convention) to its new location using Windows Explorer. The CodeWarrior IDE reconstructs the project state when you select a **Bring Up To Date** or **Make** operation. In a revision control system, you need only check in the main project file and not the data files.

If your project file references other files with absolute access paths, you might need to modify the paths when you move the project. See *Configuring access paths* on page 10-12 for more information.

3.3.7 Importing and exporting a project as XML

You can export a project file in *eXtensible Markup Language* (XML) format. This format is useful when you want to use the CodeWarrior IDE file comparison feature to compare and merge the contents of different project files. See *Comparing XML-formatted projects* on page 2-28 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.4 Working with project windows

The CodeWarrior IDE provides different ways to display project windows:

- *Multiple Document Interface (MDI)* where windows are enclosed within the main CodeWarrior IDE window. See *Projects displayed using MDI*.
- *Floating Document Interface (FDI)* where windows can be moved over the entire Desktop. See *Projects displayed using FDI* on page 3-20.

The MDI interface provides additional options to dock and float windows (similar to the FDI interface). The MDI interface is the default display option. See *Configuring other settings* on page 9-11 for information on how to change this default.

When you have selected how you want the project windows to be displayed you can save the selection as a workspace. See *Saving the current workspace or creating a new workspace* on page 3-22.

3.4.1 Projects displayed using MDI

The MDI option displays all project windows enclosed within the main CodeWarrior IDE window (Figure 3-8).

Figure 3-8 Projects displayed using MDI

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

In addition, MDI also provides you with options to *dock* the windows in the IDE application window or *float* the windows so they can be moved freely around the desktop.

To dock or float a window:

1. Right-click over the window title bar.

   A pop-up menu is displayed that provides the following choices:
   - Docked
   - Floating
   - MDI Child.

2. Select **Docked** to dock the window, or **Floating** to float the window.

Using these options you can customize the display of MDI windows to suit your particular requirements (Figure 3-9).

Figure 3-9 MDI window display options

To change a docked window to a floating window, double-click on the window title bar.

To change a floating window back to an MDI window, do either of the following:

- right-click over the window and select **MDI Child** from the menu displayed.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- drag the window into any corner or edge of the CodeWarrior IDE window. An outline of the window is displayed to indicate where the window will be docked. To dock the window, release the mouse button.

You can also stack docked windows by dragging the window on top of an existing docked window. This is a very efficient way of organizing multiple files and projects on your desktop (Figure 3-10).

Figure 3-10 Docked and stacked file and project windows

3.4.2 Projects displayed using FDI

The FDI option displays all open project windows as separate windows that can be moved around the desktop. The CodeWarrior IDE window changes to show just the main menu bar and toolbar. (see Figure 3-11 on page 3-21).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-11 Projects displayed using FDI

See *Configuring other settings* on page 9-11 for information on how to change the CodeWarrior IDE to using this type of interface.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.5 Working with project workspaces

The CodeWarrior IDE saves information about the current state of all open projects and all recently opened projects. This information is stored by CodeWarrior as a *workspace*. The workspace contains information about the size, location, and the docked state of all project windows.

The CodeWarrior IDE can use a default workspace or one that you create. The CodeWarrior IDE works with one workspace at a time. You can save and re-apply a workspace from one CodeWarrior IDE session to the next.

By default, the workspace for a project is saved whenever you close a project. When the project is opened again it will be displayed in the same state as it was when it was closed. (To change this setting, see *Configuring IDE extras* on page 9-9).

You can also create a workspace and apply this to replace the current workspace. Saving your own workspace is useful if you have opened several project windows and want to revert to your original layout and close any extraneous project windows.

3.5.1 Saving the current workspace or creating a new workspace

To save the current workspace or create a new workspace:

1. Arrange and size your project windows as required.
2. Select **Save Workspace** from the **File** menu to save your current workspace or **Save Workspace As...** from the **File** menu to create a new workspace.

   The Save As dialog box is displayed.
3. Enter a name for the current workspace or new workspace.

   The workspace file is saved with a `.cww` file extension.

3.5.2 Opening workspaces

To open a previously saved workspace:

1. Select **Open Workspace** from the **File** menu.

   The Choose a workspace file dialog box is displayed.
2. Browse to the workspace file you want to open.

   The CodeWarrior IDE opens the selected workspace and applies its settings. This might mean that current projects close automatically and new projects open.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To open a recent workspace:

1. Select **Open Recent** from the **File** menu.

   A submenu is displayed showing all recent files, projects and workspaces. A tick appears alongside the active workspace.

2. Select the workspace you want to open.

   The CodeWarrior IDE opens the selected workspace and applies its settings. This might mean that current projects close automatically and new projects open.

See *Configuring IDE extras* on page 9-9 for information on setting the number of files that the CodeWarrior IDE stores in this menu.

3.5.3 Closing workspaces

To close a workspace:

1. Select **Close Workspace** from the **File** menu.

   If you have made changes to the workspace since you opened it then an alert dialog box is displayed (Figure 3-12).

Figure 3-12 Alert dialog box

2. Click **Don't Save** to close the workspace without saving the changes you have made.

   Click **Save** to save the changes you have made to the workspace before closing the workspace.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.6 Working with project stationery

A project stationery file is typically a minimal, preconfigured template project file. You can use project stationery to create a new project quickly. When you create a new project or open a project stationery file, the CodeWarrior IDE creates a new project and, optionally, a new directory for the project. It then copies all the files related to the stationery project to the new directory.

Project stationery can include:

- preconfigured build target settings for the project
- predefined build targets, sub-projects, and build dependencies
- all files included in the project.

3.6.1 The project stationery directory

The project stationery directory is located in:
`install_directory\IDEs\CodeWarrior\CodeWarrior\...\...\win_32-pentium\Stationery`

ARM-supplied project stationery for common types of projects are located in subdirectories in this directory. You can create your own project stationery by saving preconfigured projects, together with their support files, in the project stationery directory.

3.6.2 ARM project stationery

The ARM version of the CodeWarrior IDE is supplied with a number of default stationery projects to enable you to create an ARM, Thumb, or Thumb/ARM interworking project quickly.

All the supplied stationery projects use:

- default target settings (little-endian ARM7TDMI®)
- default *Procedure Call Standard for the ARM Architecture* (AAPCS) options for the compiler and assembler
- separate build targets for Debug and Release (see *Predefined build targets* on page 4-4).

This means that you must reconfigure the target options if, for example, you want to target a different ARM processor. See Chapter 10 *Configuring a Build Target* for detailed information. For more information about using the ARM project stationery to create projects, see Chapter 4 *Working with the ARM Project Stationery*.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.7 Managing files in a project

This section describes how to manage files in your project. It provides information on:

- *Selecting files and groups*
- *Adding files to a project* on page 3-26
- *Grouping files in a project* on page 3-30
- *Moving files and groups* on page 3-32
- *Removing files and groups* on page 3-33
- *Touching and untouching files* on page 3-34
- *Examining and changing project information for a file* on page 3-36.

3.7.1 Selecting files and groups

From the project window, you can select one or more files and groups to open, compile, check syntax, remove from the project, or move to a different group. Selecting a group selects all the files in the group, regardless of whether or not the files appear to be selected.

Selection by mouse-clicking

You can use the following methods to select files with the mouse:

- To select a single file or group in the project window, click its name.
- To select a consecutive list of files or groups either:
  - Click the first file or group in the list, and then Shift+click the last file or group. Everything between and including the first and last file or group is selected.
  - Drag-select files in the same way as on the Windows desktop.
- To select non-contiguous files or groups, Ctrl+click the file and group names.

Selection by keyboard

To select an item using the keyboard, type the first few characters of the name of the item you want to select. As you type, the CodeWarrior IDE selects the file in the project that most closely matches the characters you have typed. Press the Backspace key if you make a mistake. Press the Enter key to open a file.

Note

Only files in currently expanded groups in the project window can be selected this way. Files in collapsed groups are not matched with your keystrokes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.7.2 Adding files to a project

This section describes how to add files to your project. You can use the following methods to add files:

The Add Files command

Select **Add Files…** from the **Project** menu or right-click in the project window (**Files** or **Link Order** tab) and select **Add Files…** to add one or more files to the current project. See *Using the Add Files command* on page 3-27 for details.

Drag and drop

Use drag and drop to add one or more files to the current project. See *Adding files with drag and drop* on page 3-28 for details.

The Add Window command

Select **Add Window** to add the file in the currently active editor window. See *Adding the current editor window* on page 3-29.

When you add a file to a project, the CodeWarrior IDE adds the path to that file to the project access paths, and displays a message in the message window.

Note

You can change the structure of the files in the project but this does *not* change the location of the files in the Windows filesystem. The group structure of the source files in the project is independent of the directory structure of the source files in the Windows filesystem. See *Grouping files in a project* on page 3-30 for more information.

Filename requirements

Filenames must conform to the following rules or the CodeWarrior IDE will not add them to the project:

- Filename extensions for the file type you want to add must be defined in the File Mappings configuration panel. See *Configuring file mappings* on page 10-25 for more information.
- You cannot add multiple copies of source files that generate object output, such a C, C++, or assembly language source files. You can add multiple copies of header files. The CodeWarrior IDE searches the defined search paths and uses the first file with the correct name that it locates. It does not continue to search for header files with the same name.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Where added files are displayed

When you add files to a project, they are placed either:

- after the currently selected item in the project window
- at the bottom of the project window if no item is currently selected in the project window.

To place a new file or group in a specific location, you must select the file or group above the location where you want the file to be added before you select **Add Files** or **Add Window**.

If you select a group, the added files are placed at the end of that group regardless of whether or not the group is expanded or collapsed. See *Selecting files and groups* on page 3-25 for more information on selecting files and groups of files.

Note

If you drag and drop a directory of source files onto your project window, a new group is created and appended to your project. The added files are placed in the new group.

See *Moving files and groups* on page 3-32 for information on how to move a file, or group of files, to a new location within the project.

Using the Add Files command

To add source code files, libraries, and other files to your project:

1. Select **Add Files…** from the **Project** menu or right-click in the project window (**Files** or **Link Order** tab) and select **Add Files…**. The CodeWarrior IDE displays a Select Files to add... dialog box (Figure 3-13).

Figure 3-13 Adding files to a project

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Use the **Files of type** drop-down menu to filter the types of files displayed in the dialog box.

   Note

   If you select **All Files,** the dialog box displays all files regardless of their filename extension. However, the CodeWarrior IDE will not add files that do not have a recognized filename extension set in the File Mappings configuration panel. See *Configuring file mappings* on page 10-25 for more information.

3. Change directory to the location of the files you want to add and select the files to be added:
   - To select a single file, click on its filename. Alternatively, double-click the file to add it to your project immediately.
   - To select multiple files, press Ctrl and click on the filenames in the dialog box.
   - To select a contiguous group of files, click on the first filename in the group, then press the Shift key and click on the last file in the group. Alternatively you can drag the mouse over the files you want to select.

4. Click **Add** to add the selected files. If your project contains multiple build targets, the CodeWarrior IDE displays an Add Files dialog box (Figure 3-14). Select the build targets to which you want the files added, or click **Cancel** to close the dialog box without adding any files to the project.

Figure 3-14 Add Files dialog box

Adding files with drag and drop

You can drag suitable files or directories directly to an open project window. When you drag files onto the project window, the CodeWarrior IDE verifies that the files can be added to the project. When you drag a directory, the CodeWarrior IDE checks to ensure

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

that the directory, or one of its subdirectories, contains at least one file with a recognized filename extension, and that file is not already in the project. Directories are added to the project as new groups.

If the selection does not contain at least one file recognized by the CodeWarrior IDE, the drag is not accepted. See *Configuring file mappings* on page 10-25 for more information on configuring the CodeWarrior IDE to recognize files.

To add files to your project with drag and drop:

1. Select the files or directories you want to add to the project.

   You can select files in many places, including the desktop or the multifile search list in the CodeWarrior IDE Find dialog box.

2. Drag your selection onto the project window.

3. Use the focus bar (an underline) that appears in the project window to select the location where the files will be inserted (see *The focus bar* on page 3-33).

4. Release the mouse button (drop the files) to add the dragged items to the project. The items are inserted below the position of the focus bar.

   If your project contains multiple build targets, the CodeWarrior IDE displays an Add Files dialog box (see Figure 3-14 on page 3-28). Select the build targets to which you want the files added.

Note

- You cannot drag entire volumes, such as your hard disk, onto the project window.
- You can drag files from the project window to another application to open them in that application.

See *Removing files and groups* on page 3-33 for more information on removing files from the project window.

Adding the current editor window

The **Add Window** command adds the file displayed in the active editor window to the default project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

The **Add Window** option (on the **Project** menu) is enabled when the active window is a text file, the file is not yet in the project, and the file either has a recognized filename extension, or is an unsaved window. See *Configuring file mappings* on page 10-25 for more information. The **Add Window** menu item is disabled otherwise.

To add the current editor file:

1. Select a location in the project window.
2. Open the source code file or text file in the editor.
3. Select **Add filename to project** from the **Project** menu:
   - If the editor window is untitled the CodeWarrior IDE displays the Save As dialog box. The file is added to the open project after you save it.
   - If your project contains multiple build targets, the CodeWarrior IDE displays an Add Files dialog box (see Figure 3-14 on page 3-28). Select the build targets to which you want the files added.

3.7.3 Grouping files in a project

The CodeWarrior IDE enables you to organize your source code files into groups. Groups are the CodeWarrior IDE equivalent of a Windows directory. For example, if you drag a directory of source files onto the project window, the CodeWarrior IDE creates a new group with the same name as the directory. However, CodeWarrior IDE groups are independent of the directory structure of your source files. You can create any group structure you want in the CodeWarrior IDE.

Creating groups

To create a new group:

1. Ensure that the project window is the active window, and that the Files view is selected.
2. Select a location for the new group in the project window. The CodeWarrior IDE will place the new group immediately below a selected item in the project window hierarchy. If no item is selected the CodeWarrior IDE will place the group at the top of the hierarchy.
3. Select **Create Group** from the **Project** menu. The CodeWarrior IDE displays a Create Group dialog box (see Figure 3-15 on page 3-31).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-15 Create Group dialog box

4. Enter a name for the new group and click **OK**. The CodeWarrior IDE creates the new group at the selected location.

Renaming groups

To rename a group you have already created:

1. Select the group you want to rename by clicking on it, or using the arrow keys to navigate to the group in the project window.
2. Press the Enter key, or double-click on the group in the project window. The CodeWarrior IDE displays the Rename Group dialog box (Figure 3-16).

Figure 3-16 Rename Group dialog box

Note

If you select more than one group, the CodeWarrior IDE displays the Rename Group dialog box once for each group. The Enter group name text field displays the name of the current group.

3. Enter a new name for the group and click **OK** to rename the group.

Expanding and collapsing groups

Groups display files in collapsible hierarchical lists. There are a number of ways to toggle a group list between its expanded state and its collapsed state:

- Click the hierarchical control next to the group name to toggle the display of that group only.
- Alt+click a hierarchical control to toggle the display of the group and all its subgroups. Other groups at the same level are not changed.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- Ctrl+click any hierarchical control to toggle the display of all groups at the same level.
- Ctrl+Alt+click any hierarchical control to toggle the display of all groups and subgroups (Figure 3-17).

Figure 3-17 Expanding groups and subgroups

3.7.4 Moving files and groups

To move one or more files or groups within a Files view, or to arrange build targets in a Targets view:

1. Select the files or groups to be moved. Selecting a group includes all the files in that group, regardless of whether or not those files are visually selected in the project window. See *Selecting files and groups* on page 3-25 for more information.
2. Drag the selected files or groups to their new location in the project window. The focus bar (an underline) indicates where the selected files are moved when the mouse button is released (see *The focus bar* on page 3-33).
3. Release the mouse button when the focus bar is displayed at the position you want place the files or groups. The selected files or groups are moved to the new position.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The focus bar

The focus bar has a small arrow at the left end that indicates the level of insertion into the existing hierarchy. If the arrow is to the left of a group icon, the insertion is at the same level as the target group (Figure 3-18).

Figure 3-18 Moving a file, below a group

If the arrow appears to the right of the icon, the files are inserted into the target group (Figure 3-19).

Figure 3-19 Moving a file, into a group

3.7.5 Removing files and groups

You can remove files from either the Files view or the Link Order view of the project window. A file is only removed from a project, it is not removed from your filesystem.

Note

If you remove files from the Files view, *they are removed from all the build targets in the project*. If you remove files from the Link Order view, the files are removed from the current build target only. See *Build targets* on page 3-2 for more information on build targets.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To remove one or more files or groups from the project window:

1. Click either the **Files** tab or the **Link Order** tab, depending on whether you want to remove the files from the entire project, or from the current build target only.

2. Select the files or groups you want to remove.

   Note

   Selecting a group includes all of the files in that group, regardless of whether or not those files are visually selected in the project window. See *Selecting files and groups* on page 3-25 for more information on selecting files.

3. Either:
   - press Delete
   - right-click on the selected files and select **Delete** from the context menu.

   The CodeWarrior IDE displays a confirmation dialog box (Figure 3-20).

Figure 3-20 Remove file confirmation dialog box

4. Click either:
   - **Cancel** to leave the files in the project.
   - **OK** to continue. The selected files and groups are removed from the project or the current build target.

   Note

   You cannot undo this operation. If you remove a file or a group by mistake, you must re-add the removed files by one of the methods described in *Adding files to a project* on page 3-26.

3.7.6 Touching and untouching files

The CodeWarrior IDE does not always recognize file changes, and might not recompile changed files in some cases. Use the Touch column to mark files that need to be compiled (see Figure 3-3 on page 3-5).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can touch files in the following ways:

- Click in the Touch column next to the filename in the Project Files view to toggle the touch status of the file. A check mark is displayed in the Touch column next to the filename to indicate that the file will be recompiled the next time you build your project.
- Select **Touch** from the **Header Files** pop-up menu for the file.

  Note

  If the file has not changed since it was last compiled, the first command in the **Header Files** pop-up menu is **Touch**. If the file has changed since it was last compiled, the **Untouch** command is shown.

To unmark files so that they are not compiled, click in the Touch column again, or select **Untouch** from the **Header Files** pop-up menu.

Note

You can only untouch files that have been marked for compilation with the **Touch** command. You cannot untouch files that are marked for compilation because they have been modified.

Synchronizing modification dates

To update the modification dates stored in your project file either:

- select **Synchronize Modification Dates** from the **Project** menu.
- click the check icon in the project window toolbar (Figure 3-21).

Figure 3-21 Synchronize modification dates

The CodeWarrior IDE checks the modification date for each file in the project. If the file has been modified since it was last compiled, the CodeWarrior IDE marks it for recompilation. This command is useful if you have modified source files outside the CodeWarrior IDE, for example, by using a third-party text editor.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.7.7 Examining and changing project information for a file

You can use the Project Inspector window to view and configure information for the source files in your project. The Project Inspector window consists of:

- An **Attributes** tab that displays file attributes such as the name and location of the file, the code and data size of the file, if it has been compiled, and whether or not debug information is generated for the file.
- A **Targets** tab that displays a list of the build targets that include a specific file.

To open the Project Inspector window:

1. Select the source file or library for which you want to view information in the Files view or the Link Order view of the project window.
2. Select **Project Inspector** from the **View** menu. The CodeWarrior IDE displays the Project Inspector window with the **Attributes** tab selected (Figure 3-22).

Figure 3-22 Project Inspector, Attributes

The Project Inspector window shows project information for the currently selected file or files. You change the file selection without closing the Project Inspector window.

You can use this configuration panel to specify whether debug information is generated for the current file when it is compiled. See *Generating debug information* on page 5-3 for more information on configuring debug information for files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click the **Targets** tab to display the Targets list (Figure 3-23) and select the build targets that you want to include the current file.

Figure 3-23 Project Inspector, Targets

See *Assigning files to build targets* on page 3-45 for more information on how to select files for inclusion in specific build targets.

4. Click either:
   - **Save** to save your changes
   - **Revert** to discard your changes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.8 Configuring the CodeWarrior IDE for complex or multi-user projects

The CodeWarrior IDE has a number of configuration options that can affect how, and where, project source files and libraries are searched for. If set incorrectly, these options can cause unexpected behavior for large or complex projects such as:

- projects that have multiple source files with the same name in different directories
- multi-user projects where more than one developer is modifying sources, especially if a version control system is in use
- projects that use embedded sub-projects or build targets.

The following configuration options are recommended for complex, or multi-user projects:

- Ensure that the source and library directories specified for your project are not searched recursively by configuring the Access Paths configuration panel.

  In general, for large projects recursive searching of access paths is prohibitively slow. In addition, in some circumstances the CodeWarrior IDE fails to find, or finds the wrong, dependent sub-project or source file.

  The default ARM project stationery is set to search the library and include directories non-recursively. However, if you make a new project, the CodeWarrior IDE default is to add the `{Compiler}` directories recursively to your system access paths. This means that every sub-directory of the directory in which the CodeWarrior IDE is installed is added recursively to your system paths.

  See *Setting access path options* on page 10-14 for more information on configuring recursive searching.

- Ensure that the **Always Search User Paths** option is selected in the Access Paths configuration panel.

  If this option is not selected, any source file found in a system search path is effectively promoted to an unchanging system file. This means that if a later version of the source file is placed in a user search path, it is not found by the CodeWarrior IDE until this option is selected. In particular, if your system paths are defined to be searched recursively, the CodeWarrior IDE might find unexpected versions of source files in system paths, and ignore newer versions in user paths.

  See *Setting access path options* on page 10-14 for more information on configuring the **Always Search User Paths** option.

- Ensure that the **Use Modification Date Caching** option is *not* selected in the Build Extras configuration panel. If this option is selected, the CodeWarrior IDE might not be able to determine if a source file has been modified outside the

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

immediate CodeWarrior IDE environment. This applies if you are using a third-party editor, or for multi-user development environments where source files can be modified and checked in through version control systems.

See *Configuring build extras* on page 10-21 for more information on configuring cache modification dates option.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.9 Working with multiple build targets and sub-projects

You can use the CodeWarrior IDE to create project files that use complex build rules and dependencies. This section describes how to do this.

For example, the default ARM project stationery defines separate build targets for:

- release code (Release)
- debug code (Debug).

In addition, the Thumb/ARM interworking project stationery uses separate build targets for ARM code and Thumb code, and defines build dependencies between the two build targets.

Each build target in a project has its own configuration settings. For example, the Debug build target in the ARM-supplied project stationery has code optimizations disabled. The Release build target has code optimizations enabled. (See Chapter 4 *Working with the ARM Project Stationery* for more information.)

This section describes:

- *Overview of complex projects*
- *Creating a new build target* on page 3-44
- *Assigning files to build targets* on page 3-45
- *Changing a build target name* on page 3-47
- *Assigning build target dependencies* on page 3-48
- *Building all targets in a project* on page 3-52
- *Creating sub-projects within projects* on page 3-53.

3.9.1 Overview of complex projects

Complex projects contain either multiple build targets, or multiple sub-projects, or both. You can use multiple build targets and sub-projects to create complex build relationships between parts of your code that rely on each other, or that must be compiled with different tools or build options:

Multiple build targets

A CodeWarrior IDE project can contain multiple build targets, each with its own target settings, source files, and output options. You can define build and link dependencies between build targets that enable you to link the output of multiple build targets. See *Assigning build target dependencies* on page 3-48 for detailed information on defining dependencies between multiple build targets.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

sub-projects A sub-project is a standalone project nested within another project file. You can use sub-projects to organize your build system into separate project files that can be separately maintained. For example, you can use sub-projects to build and maintain libraries that are used in your main project file.

You can link the output from any build target in a sub-project with the output object code from the main project. This means, for example, that you can link a Release build of a library, with a Debug build of your main project. See *Creating sub-projects within projects* on page 3-53 for detailed information on using sub-projects.

Strategy for creating complex projects

You can create complex projects using either, or both, multiple build targets and sub-projects. A number of factors can affect the most appropriate choice, including:

Project structure

Software development projects often consist of several sub-projects worked on by different teams. You can create CodeWarrior IDE sub-projects for team developments and use a master project to pull the sub-projects together.

The number of build targets

Projects can contain a maximum of 255 build targets. Before you reach that limit, multiple build targets will affect available memory and project load times. Projects with several build targets take up more disk space, take longer to load, and use more memory.

If your project contains more than ten to twenty build targets you can improve performance by moving some of them off to sub-projects.

Including well tested code

Any code that is not built often and uses a distinct set of source files is a good candidate for moving to a sub-project. For example, you can include a sub-project based on the ARM Object Library stationery, and link with the Release target output to include well tested library code that has been built with high optimization options. In addition, you can specify whether or not a sub-project is rebuilt when the main project is built. Dependent build targets are always rebuilt when the main build target is built.

Another good candidate for moving to a sub-project is an externally built library which is built using a different build environment but is still required as part of the main project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Including closely related code

Code that is an integral part of your main project, but requires distinct build options, is a good candidate for a dependent build target. For example, the Thumb/ARM interworking project stationery uses separate build targets for ARM and Thumb code, and defines the ARM build targets as dependents to the Thumb build targets to generate a Thumb/ARM interworking image.

Access to source code

If you want access to all your project source code from a single project file, then using multiple build targets is a good choice. sub-projects are better when you want to keep separate, standalone project files.

Some CodeWarrior IDE features are designed to work at the project level. For example, a multifile Search and Replace operation can search all source code for all build targets in the current project, but will not search source code in a sub-project unless the sub-project source files are specifically added to the search list.

Setting the current build target

If you define multiple build targets in your project, you must select the specific build target when you:

- set target options. When you set target options, the settings apply only to the currently selected build target. See Chapter 10 *Configuring a Build Target* for more information on configuring target options.
- perform a build operation by selecting the **Compile**, **Make**, or **Bring Up to Date** menu items from the **Project** menu. By default, these commands apply only to the selected build target. This means that you must perform build operations separately for each build target in your project.

  Note

  If you define build dependencies between build targets, the CodeWarrior IDE compiles all dependent build targets when you compile the main build target. You can use this, for example, to set up a build target that builds all other build targets in your project. See *Building all targets in a project* on page 3-52 for more information.

  See *Compiling and linking a project* on page 4-35 for more information on building your project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To set the active build target in a project:

From the CodeWarrior IDE menu bar

Select the build target from the **Set Default Target** menu in the **Project** menu.

From the Build Target drop-down menu

Select the build target from the **Build Target** drop-down menu (Figure 3-24).

Figure 3-24 Build Target drop-down menu

From the Targets view

The current build target is denoted by a circular icon (an archery target) with an arrow going into it (Figure 3-25). Click on an empty archery target to select a new build target. The arrow moves and the new target name is displayed.

Figure 3-25 Targets view

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3.9.2 Creating a new build target

You can create new build targets from the Targets view of the project window. To create a new build target in your project:

1. Click the **Targets** tab in your project window to display the Targets view (Figure 3-26).

Figure 3-26 Targets tab

2. Select **Create Target** from the **Project** menu.

   The CodeWarrior IDE displays a New Target dialog box (Figure 3-27).

Figure 3-27 New Target dialog box

3. Enter the name of the new build target in the Name for new target text field.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select the type of target you want to create:
   - Select the **Empty target** option if you want to create a new empty target. If you select this option you must configure all the settings of the build target as if you had created a new empty project.
   - Select the **Clone existing target** option if you want to use the settings and files from a previously defined target as a starting point for your new target. Select the target you want to clone from the pop-up list of defined targets.
5. Click **OK** to create the new build target. The new build target is added to the list of targets displayed in the Targets view.
6. Modify the new build target to suit your requirements. See:
   - *Assigning files to build targets* for information on how to include specific source and library files in the build target.
   - *Assigning build target dependencies* on page 3-48 for information on how to create build target dependencies for the new target.
   - Chapter 10 *Configuring a Build Target* for information on how to configure the target settings and options for you build target.

3.9.3 Assigning files to build targets

You can assign files to build targets using either:

- the Target column in the project window Files view (see *Including a file in a build target using the Target column*)
- the Project Inspector (see *Including a file in a build target using the Project Inspector* on page 3-46).

You can assign the same file to any number of defined build targets in a project.

Including a file in a build target using the Target column

The Target column in the project window Files view indicates whether a file is in the current build target. The CodeWarrior IDE displays this column only if the project has more than one build target. If a file is in the current build target, a dot is displayed in the Target column next to the file (see Figure 3-28 on page 3-46).

To toggle the inclusion of a file in a build target:

1. Ensure that the build target you want to assign the file to is the currently active build target. See *Setting the current build target* on page 3-42 for information on how to change the active build target.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click in the **Target** column next to the file (Figure 3-28):

Figure 3-28 Including files, Target column

Here:

- If the file is not already marked as being included in the current build, the CodeWarrior IDE places the build marker in the column to indicate that the file is included in the current build.
- If the file is already included in the current build, the CodeWarrior IDE removes the file from the build.

Note

To assign (or remove) all the files to the current build target, Alt+click in the Target column.

Including a file in a build target using the Project Inspector

You can use the Project Inspector window to assign a file to any defined build target. To use the Project Inspector:

1. Select the file you want to assign in the project window.
2. Select **Project Inspector** from the **View** menu. The CodeWarrior IDE displays the Project Inspector window.
3. Click the **Targets** tab. The CodeWarrior IDE displays a Target window for the file you selected (see Figure 3-29 on page 3-47).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-29 Including files, Project Inspector

4. Select the check box next to a build target to include the file in that build target. Deselect the check box to exclude the file from that build target.

5. Click either:
   - **Revert**, to undo the changes you have made
   - **Save**, to apply the changes.

3.9.4 Changing a build target name

This section describes how to change the name of a build target. For detailed information on setting other target options see Chapter 10 *Configuring a Build Target*.

To change the name of a build target in the Targets view of the project window:

1. Click the **Targets** tab to display the Targets view in the project window (see Figure 3-26 on page 3-44).

2. Double-click the name of the build target you want to rename. The CodeWarrior IDE displays the Target Settings window for that build target.

3. Select **Target Settings** from the list of available panels and change the name of the build target in the Target Name text field (Figure 3-30 on page 3-48).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-30 Renaming a build target

4. Click **OK** to save your changes.

You can also open the Target Settings window by selecting **Release Settings...** from the **Edit** menu. See *Configuring target settings* on page 10-8 for more information.

3.9.5 Assigning build target dependencies

You can configure a build target to depend on other build targets. Build target dependencies are useful when you want to ensure that the CodeWarrior IDE builds one or more specific build targets before the main, containing build target. In addition, you can create link dependencies between build targets. When you make your project, the CodeWarrior IDE compiles the dependent build target first, and then links its output with the output from the main build target.

For clarity, a build target that is a dependency of another build target is referred to as a *subtarget*.

Note

When you have build targets that are dependencies of another build target, they must not have the same output name as the dependent build target.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

This section describes how to set up build target dependencies. See also:

- *Creating a new build target* on page 3-44 for more information on creating build targets.
- *Setting the current build target* on page 3-42 for more information on setting the current build target.
- *Strategy for creating complex projects* on page 3-41 for information on strategies for setting up complex projects with build targets and sub-projects.

To create a new, dependent build target (sub-target) and link its output with an existing build target:

1. Open the project to which you want to add the build target and create a new build target (see *Creating a new build target* on page 3-44.)

   The new target is displayed in the list of targets (Figure 3-31). By default, the new target is not dependent on any existing targets.

Figure 3-31 Creating a new build target

2. Add the new build target as a dependent to the main build target by dragging it below and to the right of the main build target (see Figure 3-32 on page 3-50).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-32 Dragging a dependent build target

3. Click the plus sign next to the main build target to display the list of dependencies. The dependent build target is listed in italics underneath the main build target (see Figure 3-33 on page 3-51). You can add the dependent build target to as many main build targets as you require. The CodeWarrior IDE compiles the dependent build target before attempting to compile the main build target.

Note

By default, the CodeWarrior IDE does not link the output from the dependent build target with the output from the main build target. You must explicitly chain the build targets if you want to link their output.

The output file from the dependent build target is displayed in the Files view of the project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-33 Dependent build target

4. Add files to the dependent build target. When you use the **Add Files** command, you can specify the build targets to which the files are added. To add a file to the dependent build target only, deselect the main build targets in the Add Files dialog box (Figure 3-34).

Figure 3-34 Adding files to a dependent build target

5. (Optional) Click in the Link column next to the italic build target entry to chain the dependent build target to the main target if you want the CodeWarrior IDE to link the object code from the dependent build target with the main build target. A dot is displayed in the link column to indicate that the build targets are chained (see Figure 3-35 on page 3-52). This is most commonly done when the output of the sub-target is a library, partially linked object, or directory of objects.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Click the **Link Order** tab if you want to display the order in which the output objects are linked. You can drag and drop the files in the Link Order view to change the link order.

Figure 3-35 Linking output from a dependent build target

6. Click the **Make** button, or select **Make** from the **Project** menu. The CodeWarrior IDE compiles the dependent build target first, and links its output with the output from the main build target if you have chained the output as described in step 5.

3.9.6 Building all targets in a project

The CodeWarrior IDE does not have a single Build All command that will build all build targets in a project. However, you can use target dependencies to create a dummy build target that does nothing other than build all the other targets in your project.

To create a Build All Targets build target:

1. Create a new build target using the **Empty Target** option in the New Target dialog box. See *Creating a new build target* on page 3-44 for details.

2. Leave the target settings for the new build target as they are. That is, do not define a linker or other target settings for the new build target.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Drag the existing build targets underneath and to the right of the new build target listing in the Targets view to create a build dependency. See *Assigning build target dependencies* on page 3-48 for more information on creating dependent build targets.

   The Build All Targets build target displays an italicized list of dependent build targets. Figure 3-36 shows an example based on the ARM stationery. The new build target is named Build All Targets.

Figure 3-36 Building all build targets

4. Ensure that the Build All Targets build target is the current build target. See *Setting the current build target* on page 3-42 for more information.

5. Click the **Make** button, or select **Make** from the **Project** menu to build all the dependent build targets.

3.9.7 Creating sub-projects within projects

A sub-project is a standalone project nested within another project file. You can use sub-projects to organize your build into separate project files that can be separately maintained.

sub-projects are listed in the Files view and the Targets view of the project window with the other components of your project (see Figure 3-37 on page 3-54). They can be assigned to any build target in the main project. When you add a sub-project, you can select the build targets to which it belongs.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 3-37 sub-projects within a project

Double-click on the name of a sub-project to display a new project window so you can see the Files view and the Targets view for the sub-project.

You can configure your main project so that a `Make` command builds one or more build targets in a sub-project when it builds the container build target in the main project. You can also configure the main project so that it links the output from any build target in a sub-project to any build target in the main project. This means, for example, that you can link the Release build target output from a sub-project in which the code is well tested, with the Debug build target of your main development project.

This section includes:

- *Main steps required for compiling and linking a sub-project*
- *sub-projects with multiple build targets* on page 3-55.

For an example of creating a sub-project, see *Creating a project with a sub-project* on page 4-23.

Main steps required for compiling and linking a sub-project

To compile and link the output from a sub-project with your main project:

1. Add the sub-project to one or more build targets in the main project.
2. Specify which, if any, build target in the sub-project is to be built when the main project is built. By default, none of the build targets is built when the sub-project is first added.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Specify which, if any, output objects are to be linked with the output from the main project. By default, none of the build target output objects is linked when the sub-project is first added.

Each of these steps is independent of the others. For example, if you want, you can specify that a sub-project build target is built when the main project is built, but not linked with the main project.

You can also specify that the output from a sub-project build target is linked, but the sub-project build target is not built, when the main project is built. This is useful for creating build-all types of project that build a number of related projects but do not use their outputs.

sub-projects with multiple build targets

If you have multiple build targets in a sub-project, you can create link dependencies between the main project and any of the build targets in that sub-project (Figure 3-38).

Figure 3-38 Multiple build targets in sub-projects

The sub-projects are linked in the order given in the Link Order view. This means that, if you are using the ARM project stationery supplied with RVDS (see *ARM project stationery* on page 3-24), you can create a link dependency to any, or all, of the Debug and Release build targets in the sub-project, all of which might contain the same code, but are built with different optimization and debug options.

When you build your project, the ARM linker selects the first available object file that resolves the unresolved symbol it is processing. However, in the Link Order view, the output filenames for each of the build targets are identical. Therefore, select the output file in the Files view and use the Project Inspector to determine which output object is being linked.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

See Chapter 4 *Working with the ARM Project Stationery* for information on how to use the ARM project stationery provided with the CodeWarrior IDE to create simple projects, and advanced projects containing sub-projects.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 4
Working with the ARM Project Stationery

This chapter provides information on how to use the ARM® project stationery provided with the CodeWarrior IDE to create simple projects and advanced projects containing sub-projects. It is assumed that you have read Chapter 3 *Working with Projects*.

This chapter contains the following sections:

- *About the ARM Project stationery* on page 4-2
- *Working with an existing ARM project* on page 4-5
- *Using the Thumb ARM interworking stationery* on page 4-7
- *Creating binary ROM images* on page 4-13
- *Creating a project that builds an object library and ROM images* on page 4-14
- *Creating a project with a sub-project* on page 4-23
- *Creating your own project stationery* on page 4-29
- *Converting ARM projects to Thumb projects* on page 4-32
- *Converting Executable Image projects to Library projects* on page 4-34
- *Compiling and linking a project* on page 4-35
- *Processing output* on page 4-44
- *Running batch files with the Batch File Runner* on page 4-48.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.1 About the ARM Project stationery

The following ARM project stationery is supplied with the CodeWarrior IDE for RVDS:

- ARM Executable Image
- Thumb® Executable Image
- ARM Object Library
- Thumb Object Library
- Thumb ARM Interworking Image.

These stationery projects can be selected when a new project is created (see *Creating a new project* on page 3-11). The ARM stationery projects are described in the following sections.

4.1.1 ARM Executable Image

Use this stationery to build an executable ELF image from ARM code. It is configured to use:

- The ARM compiler (`armcc`) to compile all C or C++ files in ARM state.
- The ARM assembler (`armasm`) to assemble all files with a `.s` filename extension.
- The ARM linker (`armlink`) to link a simple executable ELF image.
- The RealView Debugger to debug and run executable images output by the project.

4.1.2 Thumb Executable Image

Use this stationery to build an executable ELF image from Thumb code. It is configured to use:

- The ARM compiler to compile all Thumb C or C++ files in Thumb state.
- The ARM assembler to assemble all files with a `.s` filename extension. By default, the Thumb Executable Image stationery configures the assembler to assemble for Thumb. See the *RealView Compilation Tools Assembler Guide* for information on switching the assembler to compile ARM code.
- The ARM linker to link a simple executable ELF image.
- The RealView Debugger to debug and run executable images output by the project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.1.3 ARM Object Library

Use this stationery to build a library of ARM objects. The library contains ELF object format members. This stationery is similar to the ARM Executable Image stationery. The major differences are:

- This stationery is configured to use the ARM RealView Librarian, `armar` to output an object library.
- You cannot debug or run a standalone library file until it is linked into an image.

See the *RealView Compilation Tools Linker and Utilities Guide* for more information on the ARM RealView Librarian.

4.1.4 Thumb Object Library

Use this stationery to build a library of Thumb objects. The library contains ELF object format members. This stationery is similar to the Thumb Executable Image stationery. The major differences are:

- This stationery is configured to use the ARM RealView Librarian, `armar` to output an object library.
- You cannot debug or run a standalone library file until it is linked into an image.

See the *RealView Compilation Tools Linker and Utilities Guide* for more information on the ARM RealView Librarian.

4.1.5 Thumb ARM Interworking Image

Use this stationery to build an executable ELF image from interworking ARM and Thumb code. It is configured to use:

- Separate build targets for ARM code and Thumb code. The output from the Thumb build targets is chained with the corresponding ARM build targets. See *Assigning files to build targets* on page 3-45 for information on defining build dependencies between build targets.
- The compiler to compile C and C++ code included in the ARM build targets.
- The compiler to compile Thumb C and C++ code included in the Thumb build targets.
- The assembler to assemble both ARM and Thumb assembly language source.
- The linker to link the output from the Thumb and ARM build targets into an executable ELF image.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- The AAPCS interworking option for the assembler and compiler.

4.1.6 Predefined build targets

The non-interworking project stationery files define two build targets. The interworking project stationery defines an additional two build targets to compile Thumb-targeted code. The basic build targets for each of the stationery projects are:

Debug This build target is configured to build output binaries that are fully debuggable, at the expense of optimization. This build target provides the best debug view while you are developing your code. It is also configured to output basic image information in an Error and Messages window.

Release This build target is configured to build output binaries that are fully optimized, at the expense of debug information. This build target outputs optimized code suitable for release.

The Release and Debug build targets are defined in the following ARM stationery:

- ARM Executable Image
- ARM Object Library
- Thumb Executable Image
- Thumb Object Library.

This stationery uses Debug as the default build target.

Thumb/ARM interworking

In addition, the Thumb/ARM interworking project stationery uses separate build targets for ARM code and Thumb code, and defines build dependencies between build targets. The build targets are:

- ThumbRelease
- ThumbDebug (the default build target)
- ARMRelease, a dependency of ThumbRelease
- ARMDebug, a dependency of ThumbDebug.

See *Working with multiple build targets and sub-projects* on page 3-40 for more information on using the ARM project stationery to create complex projects. See *Configuring debug and optimization* on page 10-59 for information on how to set debug and optimization options.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.2 Working with an existing ARM project

This section describes how to work with an existing CodeWarrior IDE project which has been created using ARM project stationery. It includes the following sections:

- *Opening an existing CodeWarrior project*
- *Building the image for your project*
- *Running the image in the RealView Debugger* on page 4-6.

4.2.1 Opening an existing CodeWarrior project

To open an existing CodeWarrior IDE project file:

1. Select **File → Open...** from the main menu to display the Open dialog box.
2. Locate the project file for the project you want to open.

   In this example, locate the example file for the Dhrystone project:

   `install_directory\RVDS\Examples\...\dhrystone\dhrystone.mcp`
3. Click **Open** to open the project.

If no build output directories exist for the project, the CodeWarrior IDE creates directories for them in the project directory. The output directory is the directory where the CodeWarrior IDE places the final output file it builds a target in a project. This example has two build targets, so two build output directories are created called **Debug** and **Release**.

4.2.2 Building the image for your project

To build the image for your CodeWarrior IDE project, select **Project → Make** from the main menu.

When you build an image for a particular build target, the CodeWarrior IDE places the image in the output directory for that build target. For example, if you build the RVDS example `dhrystone.axf` image for the **Debug** build target, it is placed in the directory:

`install_directory\RVDS\Examples\...\dhrystone\Debug`

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.2.3 Running the image in the RealView Debugger

To run the image for your project you must set up a connection to your target:

1. Start RealView Debugger from the Windows **Start** menu.
2. Configure and connect to your debug target as described in the RealView Debugger documentation.
3. Exit RealView Debugger, without disconnecting from your target. When you next start RealView Debugger from the CodeWarrior IDE, the connection is established and your image is loaded automatically.
4. In the CodeWarrior IDE, run the debugger using one of the following options:
   - Select **Project** → **Debug** from the main menu.
   - Select **Project** → **Run** from the main menu.

The debugger starts, establishes the connection, and loads the project image to the target. Run and debug the image using the features provided with the debugger as described in your debugger documentation.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.3 Using the Thumb ARM interworking stationery

The Thumb ARM Interworking Image stationery is an example of a complex project that uses multiple, dependent, build targets to compile ARM and Thumb code separately, and then link the output into an interworking executable image. This section includes a tutorial that shows you how to use the interworking stationery. See *Working with multiple build targets and sub-projects* on page 3-40 for a detailed description of how to use complex projects.

The Thumb ARM Interworking Image stationery is set up to create an ARM/Thumb interworking project as described in *Thumb/ARM interworking* on page 4-4.

The main steps to create and setup an ARM/Thumb interworking project are:

1. Create the project (see *Creating an ARM/Thumb interworking project* on page 4-8).
2. Add your source files to the build targets (see *Adding source files to the build targets* on page 4-9).
3. Build the image (see *Building the image* on page 4-11).
4. Run the image in the debugger (see *Running the image in the RealView Debugger* on page 4-6).

4.3.1 Before you begin

Before you begin this tutorial, create a `Tutorial` directory in a location of your choice. For example, `C:\Myprojects\Tutorial`.

The tutorial uses the source files from the RVDS interwork example project:

install_directory`\RVDS\Examples\...\interwork`

There are two sets of sources in this example project:

- Use the following source files for an image with `main()` in ARM state:
  - `armmain.c`
  - `thumbsub.c`
- Use the following source files for an image with `main()` in Thumb state
  - `thumbmain.c`
  - `armsub.c`

You are instructed when to copy these sources to the `Tutorial` directory as part of the tutorial.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.3.2 Creating an ARM/Thumb interworking project

To create the ARM/Thumb interworking project:

1. Select **File** → **New** from the main menu to display the New dialog box (Figure 4-1).

Figure 4-1 New dialog box

2. Select the **Thumb ARM Interworking Image** stationery.
3. Click the **Set...** button to display the Create New Project... dialog box, and locate the directory that you want to use to hold your new project directory, for example:

   `C:\Myprojects\Tutorial`

4. In the Create New Project... dialog box:
   a. Enter the project name `ARM Thumb Interworking` in the **Project name** field.
   b. Set the **Location** field to the correct destination directory.
5. The Project name and Location fields in the New dialog are set to the values you specified in the Create New Project... dialog box.

   Note

   The project name has the same name as the project file.

6. Click **OK** to create the project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The CodeWarrior IDE:

- Creates a project directory in the location you specified. This directory has the same name as the project name you specified.
- Sets up the project with the settings defined in the Thumb ARM Interworking Image stationery template.
- Displays the Files view for the new project (Figure 4-2).

Figure 4-2 ARM Thumb Interworking project, Files view

Note

A file with the extension `.tdv` is created as the output for each sub-target. This is an intermediate build file for that sub-target, which the CodeWarrior IDE uses in the link stage to create the final output for the main build target.

4.3.3 Adding source files to the build targets

To add the source files for the interworking project, you must:

1. Copy the source files to the project directory (see *Copying the source files to the project directory* on page 4-10).
2. Add the ARM source files to the ARM build targets (see *Adding the ARM sources* on page 4-10).
3. Add the Thumb source files to the Thumb build targets (see *Adding the Thumb sources* on page 4-11).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

When you open a project and messages are produced, the Project Messages window has focus. In order to add files to your project, you must first click the tab at the top of the project window (see Figure 4-2 on page 4-9 for an example).

For more information about other ways of adding files to a project, see *Adding files to a project* on page 3-26.

Copying the source files to the project directory

Copy the `armmain.c` and `thumbsub.c` source files from the RVDS interwork example project:

install_directory`\RVDS\Examples\...\interwork`

to the tutorial project directory, for example:

`C:\Myprojects\Tutorial\ARM Thumb Interworking`

Alternatively, you can use the `thumbmain.c` and `armsub.c` source files.

Adding the ARM sources

To add your ARM source files to the ARM build targets:

1. Select **Project → Add Files…** from the main menu and select the `armmain.c` source file from the standard file dialog box. The CodeWarrior IDE displays an Add Files dialog box.
2. Deselect the Thumb build targets and click **OK** (Figure 4-3).

Figure 4-3 Add Files dialog box

The CodeWarrior IDE adds the file to the ARM build targets only.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Adding the Thumb sources

Add your `thumbsub.c` source file to the Thumb build targets. Follow the procedure described in *Adding the ARM sources* on page 4-10, but select the Thumb build targets when you add the file.

Figure 4-4 shows an example of the Files view for the **ThumbRelease** target. The Files view shows all the files that are in the project. Files that are not in the current build target do not have a black dot in the Target column, and have a code size of `n/a`.

Figure 4-4 ARM Thumb Interworking project, Files view

4.3.4 Building the image

To build the **ThumbDebug** build target:

1. Change the current build target to **ThumbDebug**.
2. Select **Project** → **Make** from the main menu.

The CodeWarrior IDE:

1. builds the dependent **ARMDebug** build target
2. builds the **ThumbDebug** build target
3. links the output from the two build targets.

To build the other interworking targets, select the appropriate build target from the **Build Target** drop-down menu. If you select an ARM build target, only the ARM part of your project source is built.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

When you build an image for a particular build target, the CodeWarrior IDE places the image in the subdirectory for that build target. In this example, the image is placed in the directory:

```
...\ARM Thumb Interworking\ThumbDebug\ARM Thumb Interworking.axf
```

Changing the build location

If you want to change the build location:

1. Select **Edit** → ***TargetName*** **Settings...** from the main menu to display the Target Settings Panels window.
2. In the Target Settings configuration panel, click the **Choose...** button to display the Please Select an Access Path dialog box.
3. Locate the directory where you want the project image to be built, then click **OK**.

   The output directory path is shown relative to the project directory. For example, if you have created an `images` subdirectory in your project directory, the path is:

   ```
   {Project}\images
   ```


Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.4 Creating binary ROM images

The default ARM project stationery provided with the CodeWarrior IDE is configured to generate a semihosted `fromelf` executable image. To convert a project to produce a simple binary image suitable for embedding in ROM you must:

1. Configure the Target Settings configuration panel to call `fromelf` as a post-linker. This is the default setting.
2. Configure the RealView FromELF configuration panel to convert the executable ELF image output by the linker to the binary format of your choice. The default output format is plain binary.
3. Configure the RealView Linker configuration panel to create the image structure you require, or use a scatter-loading description file to specify your image structure. For more information, see:
   - the example ROM projects in:
     install_directory`\RVDS\Examples\...\emb_sw_dev`
   - the chapter that describes embedded software development in the *RealView Compilation Tools Developer Guide*.

For more information about how you can configure linker options, see *Configuring linker settings* on page 10-65.

Also, see the tutorial *Creating a project that builds an object library and ROM images* on page 4-14.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.5 Creating a project that builds an object library and ROM images

This section includes a tutorial that shows you how to create a CodeWarrior IDE project that:

1. Builds an object library.
2. Builds an executable image from the object library and other source files.
3. Converts the executable image to ROM images.

It includes the following sections:

- *Summary of the CodeWarrior IDE project*
- *Creating the CodeWarrior IDE project* on page 4-15
- *Setting up the build target for the library* on page 4-16
- *Creating the application build target* on page 4-18
- *Assigning the build target dependencies* on page 4-19
- *Adding the sources to the appropriate build targets* on page 4-20
- *Creating the application.bin ROM image* on page 4-22
- *Creating the application.m32 ROM image* on page 4-22.

4.5.1 Summary of the CodeWarrior IDE project

This tutorial uses the source files from the RVDS VFP example project:

`install_directory\RVDS\Examples\...\vfpsupport\Build4`

If you examine the build options in `build.bat` or `Make` file, you see that two build targets are required to build the entire project. The build targets are:

application To compile and assemble the main project sources, and to create the `application.bin` Flash image.

You must modify this to create the `application.m32` Flash image, as described in *Creating the application.m32 ROM image* on page 4-22.

vfp_support To compile and assemble the library sources, and to create the `vfp_support.a` object library.

Note

When you have build targets that are dependencies of another build target, they must not have the same output name as the dependent build target. Therefore, you might want to use the project name as the Output Name for a build target, and specify a unique Output Name for each dependency of that build target. For clarity, a build target that is a dependency of another build target is referred to as a sub-target.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The corresponding build target name has been used as the output name in this tutorial, but you can use any unique name.

4.5.2 Creating the CodeWarrior IDE project

Before you start the tutorial, make sure that there are no projects open in the CodeWarrior IDE.

To create the project:

1. Copy the `vfpsupport` example project from the RVDS examples to a location of your choice, for example, `c:\Myprojects\Tutorial`. This copy is used for the rest of the tutorial, and is referred to as:

   `...\vfpsupport`

   If you want, you can remove the `build1`, `build2`, `build3`, `build5`, `build6`, `build7`, and `util` subdirectories from your copy.

   You might also want to remove any files that are not required, such as `*.bat`, the `.pdf` file, and `Make` file.

2. Select **File** → **New** from the main menu to display the New dialog box (see Figure 4-1 on page 4-8).

3. Select the **ARM Executable Image** stationery.

4. Click **Set...** to display the Create New Project... dialog box.

5. Locate your `...\vfpsupport` directory.

   Note

   If you want to use this project to create stationery, you must create the project file in your `...\vfpsupport` directory, rather then the `build4` subdirectory (see *Creating your own project stationery* on page 4-29 for more details).

6. Enter the project name, `vfpsupport`.

7. Deselect **Create Folder**.

8. Click **Save** to close the Create New Project... dialog box and display the New dialog box updated with your entries (see Figure 4-5 on page 4-16).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 4-5 New project, vfpsupport

9. Click **OK**. The project file (`vfpsupport.mcp`) and the `Debug`, `Release`, and `vfpsupport_Data` directories are created.

4.5.3 Setting up the build target for the library

This project uses a build target to build the `vfp_support.a` library. To set up the build target for the library:

1. Click the **Targets** tab.
2. Double-click on the **Release** build target to display the Target Settings window.
3. Select **Target Settings** in the Target Settings Panels list:
   a. Change the Target Name to `vfp_support`.
   b. Change the Linker to **ARM RealView Librarian**.
   c. Change the Post-linker to **None**.
   d. For the output directory, click **Choose...**, to display the **Please Select an Access Path** dialog box.
   e. Select the `vfp_support` subdirectory, then click **OK**. The output directory changes to `{Project}vfp_support` (see Figure 4-6 on page 4-17).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 4-6 Release Settings, vfp_support

f. Click **Apply** to apply the settings.

The build target name changes to `vfp_support`.

4. Select **RealView Target** in the Target Settings Panels list:
   a. Uncheck **Use project name**.
   b. Enter `vfp_support` as the Output Name.
   c. Click **Apply** to apply the settings.
5. Select **RealView Assembler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Set the Floating Point to **VFPv2**.
   c. Set the Floating Point Models to **Full IEEE**.
   d. Click **Apply** to apply the settings.
6. Select **RealView Compiler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Set the Floating Point to **VFPv2**.
   c. Set the Floating Point Models to **Full IEEE**.
   d. Click the **Debug / Opt** tab.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

   e. Set the Optimization Level to **2 (high optimization, restricted debug view)**.
   f. Click **Apply** to apply the settings.

7. Click **OK** to save the settings and close the dialog box.
8. Click and drag the **vfp_support** build target so that it appears first in the Targets view.

4.5.4 Creating the application build target

To create the **application** build target:

1. Double-click on the **Debug** target to display the Target Settings window.
2. Select **Target Settings** in the Target Settings Panels list:
   a. Enter application as the Target Name.
   b. Set Linker to **RealView Linker**.
   c. Make sure Post-linker is set to **RealView FromELF**.
   d. Click **Apply** to apply the settings.
3. Select **RealView Target** in the Target Settings Panels list:
   a. Uncheck **Use project name**.
   b. Enter application as the Output Name.
   c. Set the Output Type to **Linker Output**.
   d. Click **Apply** to apply the settings.
4. Select **RealView Assembler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Set the Floating Point to **VFPv2**.
   c. Set the Floating Point Models to **Full IEEE**.
   d. Click **Apply** to apply the settings.
5. Select **RealView Compiler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Click the **Debug / Opt** tab.
   c. Set the Optimization Level to **2 (high optimization, restricted debug view)**.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

d. Click the **Preprocessor** tab.

e. Enter `USE_SERIAL_PORT` in the text box. Then click **Add** so that it appears in the list of `#DEFINE` macro definitions.

f. Click **Apply** to apply the settings.

6. Select **RealView Linker** in the Target Settings Panels list:

   a. Select **Scattered** for the Linktype.

   b. Enter `build4\scatter.txt` in the Scatter description file field.

   c. Click the **Options** tab.

   d. Enter `__main` as the Image entry point.

   e. Click **Apply** to apply the settings.

   This builds the `build4\application.axf` image.

7. Select **RealView FromELF** in the Target Settings Panels list:

   a. Make sure that the Output format is **Plain binary** (the default setting).

   b. Enter `application.bin` as the Output filename.

   c. Click **Apply** to apply the settings.

   This converts the `Debug\application.axf` image to the `application.bin` ROM image.

8. Click **OK** to save the settings and close the dialog box.

4.5.5 Assigning the build target dependencies

To assign the build targets that are to be sub-targets of the **application** build target:

1. Click the **Targets** tab to display the Targets view for the project.

2. Drag and drop the **vfp_support** build target to **application**. A sub-target is created under **application**.

3. Expand the **application** build target.

4. For the **vfp_support** sub-target, click the link column so that a black dot appears opposite the sub-target.

5. Set the **application** build target to be the current build target. See Figure 4-7 on page 4-20.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 4-7 Assigning build target dependencies

4.5.6 Adding the sources to the appropriate build targets

Table 4-1 shows the source files that must be added to each build target.

Table 4-1 Source files belonging to each build target

Build target	Source files	Location
vfp_support	`*.c` `*.s`	`...\vfpsupport\vfp_support`
application	`main.c`	`...\vfpsupport`
	`*.c` `*.s`	`...\vfpsupport\build4`
	`*.c` `*.s`	`...\vfpsupport\vfp_init`

To add the source files to the appropriate build target:

1. Select **Project** → **Add Files...** from the main menu. The Select files to add... dialog box is displayed.
2. Locate the source file or files you want to add.

   To select multiple files, Ctrl+click each file.
3. Click **Open**.
4. Select the build target to add the file or files. Make sure you deselect all the other build targets.
5. Click **OK**.
6. Repeat these steps for each location listed in Table 4-1.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Confirming the files are added to the correct build targets

To confirm that you have added the files to the correct build target, repeat the following steps for each build target:

1. Click the **Files** tab to display the Files view for the project.
2. Set a build target to be the current build target.
3. Check that the file or files you have added to the current build target (see Table 4-1 on page 4-20):
   - have a marker (red tick) in the Touch column
   - do not have `n/a` in the Code and Data columns.

   Only these sources are built directly for the current build target. The remaining source files are built when the associated sub-targets are built. See Figure 4-8.

Figure 4-8 Files view, vfp_support

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.5.7 Creating the application.bin ROM image

To create the `application.bin` ROM image:

1. Click the **Targets** tab to display the Targets view for the project.
2. Set **application** to be the current build target.
3. Select **Project** → **Make** from the main menu.

The ROM image is built in the main `...\vfpsupport` project directory.

4.5.8 Creating the application.m32 ROM image

To create the `application.m32` ROM image:

1. Double-click on the **application** target to display the Target Settings window.
2. Select **RealView FromELF** in the Target Settings Panels list:
   a. Select **Motorola 32 bit Hex** as the Output format.
   b. Enter `application.m32` as the Output filename.
   c. Click **Apply** to apply the settings.
   d. Click **OK** when you are told that the target must be relinked.

   This converts the `Debug\application.axf` image to the `application.m32` ROM image.
3. Click **OK** to save the settings and close the dialog box.
4. Select **Project** → **Make** from the main menu.

The ROM image is built in the main `...\vfpsupport` project directory.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.6 Creating a project with a sub-project

The following example shows how to modify the project created in *Creating a project that builds an object library and ROM images* on page 4-14 so that it includes a sub-project. To do this, you must:

1. Create a new library project to build the `vfp_support.a` object library (see *Creating the library project*).
2. Set up the build target for the library project (see *Setting up the build target for the library* on page 4-24).
3. Add the sources to the build target for the new library project (see *Adding the sources to the build target* on page 4-25).
4. Remove the **vfp_support** build target, and associated source files, from the **vfpsupport** project (see *Removing the vfp_support build target from the main project* on page 4-25).
5. Add the library project as a sub-project to the **vfpsupport** project (see *Adding a project as a sub-project* on page 4-26).

The example also gives details on how to specify link and build dependencies.

4.6.1 Creating the library project

Before you start the tutorial, make sure that there are no projects open in the CodeWarrior IDE.

To create the library project:

1. Select **File** → **New** from the main menu.
2. Select the **ARM Object Library** stationery.
3. Click **Set...** to display the Create New Project... dialog box.
4. Locate your `...\vfpsupport\vfp_support` directory.
5. Enter `vfp_support` as the required filename, and deselect **Create Folder**.
6. Click **Save**.
7. Click **OK** to save the settings and close the dialog box.

   The `vfp_support.mcp` project file is created in `...\vfpsupport\vfp_support`, alongside the directories `vfp_support_Data`, `Release`, and `Debug`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.6.2 Setting up the build target for the library

To set up the build target for the object library project:

1. Click the **Targets** tab to display the Targets view for the project.
2. Double-click on the **Release** build target to display the Target Settings window.
3. Select **Target Settings** in the Target Settings Panels list:
   a. Change the Target Name to `vfp_support`.
   b. Change the Linker to **ARM RealView Librarian**.
   c. Make sure Post-linker is set to **None**.
   d. Click **Apply** to apply the settings.

   The build target name changes to `vfp_support`.
4. Select **RealView Assembler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Set the Floating Point to **VFPv2**.
   c. Set the Floating Point Models to **Full IEEE**.
   d. Click **Apply** to apply the settings.
5. Select **RealView Compiler** in the Target Settings Panels list:
   a. Set the Architecture or Processor to a processor with a VFP, for example, an ARM1136JF-S.
   b. Set the Floating Point to **VFPv2**.
   c. Click the **Debug / Opt** tab.
   d. Set the Optimization Level to **2 (high optimization, restricted debug view)**.
   e. Set the Floating Point Models to **Full IEEE**.
   f. Click **Apply** to apply the settings.
6. Click **OK** to save the settings and close the dialog box.
7. Right-click on the **Debug** build target, and select **Delete** from the context menu.
8. Confirm the removal of the **Debug** build target.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.6.3 Adding the sources to the build target

You must now add the source files to the build target as follows:

1. Select **Project → Add Files...** from the main menu.
2. Locate your `...\vfpsupport\vfp_support` directory.
3. Press the Ctrl key, and select all the `*.c` and `*.s` source files.
4. Click **Open**.

   Because there is only one build target for the project, all sources are automatically added to this build target.

4.6.4 Removing the vfp_support build target from the main project

To remove the **vfp_support** build target from the **vfpsupport** project:

1. Close the new **vfp_support** library project.
2. Open the main **vfpsupport** project, that is `...\vfpsupport\vfpsupport.mcp`.
3. Click the **Files** tab to display the Files view for the project.
4. Change the current build target to be **vfp_support** to see a list of the source files.

   Those files that do not have `n/a` in the Code and Data columns are part of the **vfp_support** build target (see Table 4-1 on page 4-20 and Figure 4-8 on page 4-21).
5. Select all the source files for the **vfp_support** build target. Use the Shift key or Ctrl key if necessary.
6. Right-click on the selected source files, and select **Remove** from the context menu.
7. Confirm the removal of the files.
8. Click the **Targets** tab to display the Targets view for the project.
9. Right-click on the **vfp_support** sub-target, and select **Delete** from the context menu to delete it.
10. Click **OK** when prompted.

    The build target is removed from the project. There is now only one target in the project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.6.5 Adding a project as a sub-project

To add the **vfp_support** library project as a sub-project to the **vfpsupport** project:

1. Add the sub-project to the main project:
   a. Select **Project → Add Files…** from the main menu to display the Select files to add... dialog box, shown in Figure 4-9.

Figure 4-9 Adding a sub-project

   b. Locate the `...\vfpsupport\vfp_support` directory.
   c. Select the `vfp_support.mcp` project file.
   d. Click **Open**.

   Because there is only one build target for the project, CodeWarrior automatically adds the library sub-project to the **application** build target. Figure 4-10 shows an example of the Files view for the project.

Figure 4-10 Project with sub-project, Files view

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click the **Targets** tab to display the Targets view for the project. Click the plus sign next to the build target to expand the hierarchy. Each build target in the sub-project is listed in the hierarchy (Figure 4-11).

Figure 4-11 Project with sub-project, Targets view

3. Click on the circular icon (an archery target) to select the **vfp_support** sub-project build target. The CodeWarrior IDE changes the icon so that it shows an arrow going into it (Figure 4-12). sub-targets marked in this way are built when the main project is built.

Figure 4-12 Selecting sub-project build targets for building

4. Select **Project** → **Make** from the main menu.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

When the main project is built, selected sub-targets are built first if they have changed, or have been touched. If a sub-target is not selected, it is ignored when the main project is built.

In Figure 4-12 on page 4-27, the **vfp_support** sub-target in the example sub-project `vfp_support.mcp` is built *before* the **application** build target in the main project `vfpsupport.mcp`.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.7 Creating your own project stationery

You can create your own CodeWarrior for RVDS stationery project file. Project stationery can include:

- preconfigured build target settings for the project
- predefined build targets, sub-projects, and build dependencies
- all files included in the project.

You might want to do this, for example, where multiple developers, working on different parts of a project, want to use common configuration settings.

The easiest way to create custom stationery is to modify a project based on existing stationery and then save it under a new name in the stationery directory. The CodeWarrior IDE duplicates the stationery project file and its source files when you create a new project and select your stationery project in the New dialog box.

Before you create your own project stationery, it is recommended that you familiarize yourself with the project stationery supplied by ARM Limited. See *About the ARM Project stationery* on page 4-2 for more information.

Note

You must not create a project directly in the stationery directory. Instead, create a new project in a user defined directory and then save it under a new name in the stationery directory (see *Rules for creating project stationery* for naming conventions).

4.7.1 Rules for creating project stationery

When creating project stationery, follow these rules:

- You must create a new directory for your project stationery, and give it a suitable name.
- The project file for your new stationery must be placed in the top-level directory of that stationery, with a suitable name (the directory name and project name do not have to match).
- If your sources are located in different subdirectories, make sure:
  - your project is created with the project file in the top-level directory of the project.
  - you duplicate the project directory structure in the project stationery directory.
- You do not have to copy or duplicate any `ProjectName_Data` directories in the stationery directory.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.7.2 Creating project stationery

This example creates custom stationery by modifying a project based on existing stationery and then saves it under a new name in the stationery directory. This example uses the **vfpsupport** project described in *Creating a project that builds an object library and ROM images* on page 4-14. However, you can open another project or create a new project (see, for example, *Creating an ARM/Thumb interworking project* on page 4-8).

To create your own custom stationery:

1. Open the base project (`...\vfpsupport\vfpsupport.mcp`) and modify the project settings to suit your requirements. You can add and remove files as necessary to create the base project you want.
2. Select **Save A Copy As…** from the **File** menu to display the Save a copy of project as: dialog box.
3. Locate the `Stationery` directory:

   `...\IDEs\CodeWarrior\CodeWarrior\...\...\win_32-pentium\Stationery`
4. Create a new directory for your stationery:
   a. Click the **Create New Folder** button on the dialog box.
   b. Enter a name for your new stationery directory, for example, `VFPSupport`.
   c. Double-click on the new directory to make it the current directory. The directory name appears in the Save in field.
5. Enter the name you want to use for the project stationery file in the Object name field, for example `vfpsupport.mcp`. This does not have to be the same name as the stationery directory.
6. Click **Save** to save the file.
7. Copy the project source files to the project stationery directory so that they are copied to new projects created with the project stationery.

   Note

   If your sources are located in different subdirectories of your project, you must duplicate the directory structure (see *Rules for creating project stationery* on page 4-29).

   For the **vfpsupport** project:

   a. Copy the following directories from the `...\vfpsupport` directory to the new `VFPSupport` stationery directory:
      - `build4`

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- `vfp_init`
- `vfp_support`.

Note

You do not have to copy the `Debug` and `Release` directories, because these are automatically created when you create a new project using this stationery. This is because the directories are specified in the output directory setting on the Target Settings configuration panel of the build targets.

b. Copy the `main.c` file from the `...\vfpsupport` directory to the new `VFPSupport` stationery directory.

c. Remove the following files from the `VFPSupport\build4` stationery directory:
   - `build.bat`
   - `make` file.

d. Remove the following files from the `VFPSupport\vfp_support` stationery directory:
   - `build.bat`
   - `make` file
   - `vfp_support.a` (created in *Creating a project with a sub-project* on page 4-23).

Your new project stationery is now ready to use. You can select your custom project stationery from the New dialog box when you create a new project. The project settings, source files, and any subdirectories in your project stationery are used to create the new project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.8 Converting ARM projects to Thumb projects

This example describes how to convert a **Release** build target from ARM to Thumb.

Note

To convert an existing ARM project to Thumb, you must change the following configuration options for *every* build target in the project.

To convert an ARM project to Thumb:

1. Open the ARM project you want to convert, for example, `dhrystone.mcp`.
2. Click the **Targets** tab to display the Targets view for the project.
3. Double-click on the **Release** build target to display the Target Settings window.
4. Select **RealView Compiler** in the Target Settings Panels list to display the RealView Compiler configuration panel.
   a. Select **Thumb** as the Initial State (Figure 4-13).

Figure 4-13 RealView Compiler Target configuration panel

   b. Click the **PCS / Code Gen** tab.
   c. Select **ARM/Thumb interworking** in the Procedure Call Standard Options group.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

In general, you must select interworking for any code that is directly called from code running in the other state (either ARM or Thumb). If you are not sure whether your Thumb code will be called from ARM code, you should select the interworking option.

d. Click **Apply** to apply the changes.

5. Click **RealView Assembler** in the Target Settings Panels list to display the RealView Assembler configuration panel.

   a. Click the **PCS** tab.

   b. Select **ARM / Thumb Interworking** in the Procedure Call Standard Options group (Figure 4-14).

Figure 4-14 RealView Assembler PCS configuration panel

In general you must select interworking for any code that is directly called from code running in the other state (either ARM or Thumb). If you are not sure whether your Thumb code will be called from ARM code, you should select the interworking option.

   c. Ensure that the other language settings for the assembler are appropriate for your project.

   d. Click **Apply** to apply your changes.

6. Click **OK**.

Repeat all the above steps for each build target in the project, then rebuild your project. The CodeWarrior IDE uses the ARM compiler to build your code for Thumb.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.9 Converting Executable Image projects to Library projects

This example describes how to convert an Executable Image project to a Library project.

Note

To convert an existing project to a Library project, you must change the following configuration options for *every* build target in the project.

To convert an Executable Image project to a Library project:

1. Open the project you want to convert, for example, `dhrystone.mcp`.
2. Double-click on the **Release** build target to display the Target Settings window.
3. Select **Target Settings** in the Target Settings Panels list.
4. Click the **Linker** drop-down menu and select **ARM RealView Librarian** (Figure 4-15).

Figure 4-15 Selecting the ARM RealView Librarian

5. Click **Apply** to apply your changes, then **OK**.

Repeat all the above steps for each build target in the project, then rebuild your project. The CodeWarrior IDE calls the ARM RealView Librarian to create an object library.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.10 Compiling and linking a project

The CodeWarrior IDE provides a number of ways to compile and link a project. All compiling and linking commands are available from the **Project** menu. Depending on your project type, some of these commands might be disabled or renamed. Also, a menu item might be disabled because the CodeWarrior IDE is executing another command.

If you have multiple projects open at the same time, you can set the default project that the CodeWarrior IDE uses. See *Choosing a default project* on page 3-16 for more information.

This section describes:

- *Overview of compiling and linking*
- *Compiling files* on page 4-37
- *Making a project* on page 4-41
- *Removing objects from a project* on page 4-42.

4.10.1 Overview of compiling and linking

This section assumes you are familiar with how to create a project, add source files and libraries, group your files, and set the project and build target options. You must also be familiar with features such as moving files in the project window, the project window columns, and project window drop-down menus. See Chapter 3 *Working with Projects* for more information.

Note

- The CodeWarrior IDE can only compile and link files that belong to an open project. You must have a project open before trying to compile its files.
- The **Check Syntax** command uses the compiler for the default build target to check the syntax of source files that are not in a project.

By default, all C and C++ source files are compiled using the RealView Compiler in ARM or Thumb state according to the project stationery used to create the project:

- If a project is created using the ARM Executable image project stationery then all C and C++ source files are compiled using the RealView Compiler in ARM state. The compiler switches to Thumb state for C and C++ source files with a `.tc` and `.tcpp` extension
- If a project is created using the Thumb Executable image project stationery then all C and C++ source files are compiled using the RealView Compiler in Thumb state. The compiler will switch to ARM state for C and C++ source files with a `.ac` and `.acpp` extension.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Changing the default behavior

To change this behavior, display the RealView Compiler configuration panel, select the **Target** tab, and edit the Initial State option. For example, you can force the compiler to compile all C and C++ sources using the RealView Compiler in Thumb state, making no adjustments for any ARM filename extensions.

To force all C and C++ sources to be compiled as C only or C++ only, display the RealView Compiler configuration pane, select the **Source** tab, and set the **Source Language** option. The options available are:

- ISO Standard C (1990)
- ISO Standard C++ (1998).

See *Configuring the RealView Compiler* on page 10-50 for more information.

See the *RealView Compilation Tools Compiler User Guide* for information on the RealView Compiler default behavior.

Selecting a build target

When you compile one or more files in a project, the CodeWarrior IDE compiles the files only for the currently selected build target. For example, if your current build target is the Debug build target and you recompile your source files, the object code for the Release build target is not updated, and the CodeWarrior IDE does not show the files in those build targets as being up to date. See *Setting the current build target* on page 3-42 for more information.

Note

If you want to compile all the build targets in a project with a single command you can create a master build target that includes all your other build targets as dependents. See *Assigning build target dependencies* on page 3-48 for more information.

Output file naming conventions and locations

When you compile an individual file, or make a project, the CodeWarrior IDE gives conventional names to your output objects and images. By default, project output is stored in subdirectories of the project `*_Data` directory. You can change the output location by setting the default output directory in the Target Settings configuration panel. See *Configuring target settings* on page 10-8 for more information.

Table 4-2 on page 4-37 describes the output file naming conventions and default locations.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table 4-2 Default output names and locations

Output	Naming convention	Default location in the project directory
Executable ELF image	*ProjectName*.axf	*TargetName*
Partially linked ELF object	*ProjectName*.o	*TargetName*
ARM library	*ProjectName*.a	*TargetName*
Object code	*filename*.o	*ProjectName*_Data*TargetName*\ObjectCode

4.10.2 Compiling files

This section describes how to use the CodeWarrior IDE to compile one or more source files without invoking the linker to link the files.

See *Making a project* on page 4-41 for information on compiling and linking your source files and libraries.

You can use the CodeWarrior IDE to compile:

- the file in the current editor window
- one or more selected files in a project
- all the files in a project.

The object files generated by the compilation are placed in a data subdirectory of your main project directory.

Note

The CodeWarrior IDE compiles the files only for the currently selected build target. See *Setting the current build target* on page 3-42 for more information.

The CodeWarrior IDE provides feedback on the progress of a compilation. When you compile source code files and libraries, the CodeWarrior IDE:

- places an animated build icon in the project window Touch column next to the file currently being compiled
- displays the Build Progress window showing the name of the file currently being compiled (see Figure 4-16 on page 4-38).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 4-16 Build Progress window

Compiling the current editor window

To compile a single file that is open in an editor window:

1. Ensure that the file you want to compile is part of a currently open project.
2. Click on the editor window to make it the currently active window.
3. Select **Compile** from the **Project** menu.

   Note

   The **Compile** menu item is unavailable if:

   - the active editor window does not have a source code filename extension
   - the source code file for the active editor window is not included in your project.

Compiling selected files from the project window

You can use the project window to compile one or more selected files, whether or not those files are open in an editor window. To compile source files from the project window:

1. Open the project that contains the files you want to compile.
2. Select one or more source files. See *Selecting files and groups* on page 3-25 for information on selecting multiple files in the project window.
3. Select **Compile** from the **Project** menu. The CodeWarrior IDE compiles the files you have selected regardless of whether they have been changed since the last compilation.

   Note

   The **Compile** menu item is unavailable if there is no open project.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Bringing a project up to date

When you have many newly added, modified, or touched files in your project, you can use the **Bring Up To Date** command to compile all the files. This command only runs the appropriate compiler or the assembler, it does not invoke the linker.

To bring a project up to date:

1. Ensure that the project window for the project you want to bring up to date is the active window.
2. Select **Bring Up To Date** from the **Project** menu. Source files in the project are compiled if:
   - the source file is new to the project and has not previously been compiled
   - you have changed the file since the last compilation
   - you have used the **Touch** command to mark a file for recompilation.

Note

The CodeWarrior IDE compiles files only for the currently selected build target. See *Setting the current build target* on page 3-42 for more information.

Recompiling files after making changes

The CodeWarrior IDE does not always recognize file changes and might not automatically recompile a file. For example, if you modify a file with a third-party editor and you have the **Use modification date caching** option selected in the Build Extras configuration panel, the CodeWarrior IDE will not recognize that the file has been modified. To force the CodeWarrior IDE to recompile a changed file:

1. Click on the **Header Files** pop-up menu for the file you want to recompile, and select **Touch**. See *Touching and untouching files* on page 3-34 for more information on touching files.
2. Select either **Bring Up To Date** or **Make** from the **Project** menu to recompile the files you have touched.

Note

To update the modification dates stored in the project file for all files in your project, select **Synchronize Modification Dates** from the **Project** menu. See *Synchronizing modification dates* on page 3-35 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Preprocessing source code

You can preprocess a file if you want to see what the code looks like just before compilation. The preprocessor prepares source code for the compiler by:

- interpreting directives beginning with the `#` and `$` symbols, such as `#define`, `#include`, and `#ifdef`
- removing C and C++ style comments. Comments are any text enclosed in `/* */`, or any line prefixed with `//`.

To preprocess a C or C++ source file:

1. Open the file you want to preprocess, or select the file in the currently open project window.
2. Select **Preprocess** from the **Project** menu. The preprocessed source file is displayed in a new editor window with the name *`filename.i`*.
3. (Optional) Select one of the **Save...** commands from the **File** menu to save the contents of the window to a file.

Checking syntax

You can check the syntax of your source code without compiling output objects. You can check the syntax of any source file, regardless of whether it is included in a project. However, you must have a project file open in order to check the syntax of source files that are not in a project, because the **Check Syntax** command uses the compiler defined for the current default build target. To check the syntax of a source file:

1. Select the source files to be checked. Either:
   - select one or more source files in the project window
   - open a source file in the editor and ensure that the editor window is the currently active window.
2. Select **Check Syntax** from the **Project** menu. The CodeWarrior IDE invokes the compiler for the current build target to check the syntax of the selected files. Syntax errors are reported in an Errors & Warnings window. (See *Using the message window* on page 5-9 for more information.)

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.10.3 Making a project

Select **Make** from the **Project** menu, or click the **Make** button in the project window toolbar, to compile and link your source. This command builds the project by:

- compiling newly added, modified, and touched source files to produce ELF object files
- linking object files and libraries to produce an ELF image file, or a partially linked object, or a library (using the ARM RealView Librarian)
- performing any post-link operations that you have defined for your build target, such as calling `fromelf` to convert an ELF image file to another format.

If the project has already been compiled using **Bring Up To Date** or another command, then the **Make** command performs only the link and post-link steps.

Setting the link order

You can specify the order in which files are compiled and linked using the Link Order view of the project window. By rearranging the order of the files you can resolve link errors caused by file dependencies. To set the link order:

1. Click the **Link Order** tab in the project window to see the Link Order view (Figure 4-17).

Figure 4-17 Link Order view

2. Drag files into the correct link order. Use drag and drop to reposition the files into the build order you require.

The next time you select **Bring Up To Date**, **Make**, **Run**, or **Debug**, the new build order is used when compiling the project files.

See *Link Order view* on page 3-8 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

- Changing the order of files in the Link Order view can change the order in which the object code is placed in the final binary output produced from your project. The CodeWarrior IDE invokes the RealView Linker with a list of object files in the order in which they are compiled.

  By default, the RealView Linker links object files in the order in which they are presented. You can change the linker behavior by explicitly placing output sections first or last in an image, or by using a scatter-load description file to specify the output image structure. See *Configuring the RealView Linker* on page 10-66 and the *RealView Compilation Tools Linker and Utilities Guide* for more information.

- You can generate link information by selecting options in the RealView Linker configuration panel and remaking your project. See *Configuring the RealView Linker* on page 10-66 for more information.

4.10.4 Removing objects from a project

When you compile your project, the CodeWarrior IDE saves the object code generated by the compiler and assembler in the `ProjectName_data` directory. The object code increases the size of the project directory. The **Remove Object Code...** command (**Project** menu) removes object code from a specific target, or from all targets.

Caution

To ensure that you delete the correct object code files, do not delete the contents of the `*_data` directory manually.

Removing object code

To remove the object code from a project:

1. Ensure that the project window for the project is the current window, or that the project from which you want to remove object code is selected as the current default project.

   Note

   If you remove object code while an editor window is active, the CodeWarrior IDE removes object code from the current default project, regardless of whether the file displayed in the current editor window belongs to that project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select **Remove Object Code...** from the **Project** menu to display the Remove Object Code dialog box (Figure 4-18).

Figure 4-18 Remove Object Code dialog box

3. Select the object code removal options:

   Recurse Subtargets and subprojects

   Removes object code from all sub-targets and sub-projects that belong to the parent project.

   If there are no sub-targets or sub-projects this option is grayed out.

   Compact targets

   Removes extraneous project data, for example:

   - Target data files (`.tdt` files)
   - Browser data
   - Dependency information.

4. Click the target option, either:

   All Targets

   Removes all object code data for all build targets in the project. Resets the Code and Data size of each file in the project window to zero.

   If there is only one build target this option is grayed out.

   Current Target

   Removes the objects for the current build target only. See *Setting the current build target* on page 3-42 for more information on changing the build target.

5. Click **Cancel** to cancel the operation so that object code is not removed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.11 Processing output

This section describes how to process project output. It describes:

- *Disassembling code*
- *Converting output ELF images to other formats* on page 4-45
- *Disassembling output ELF images* on page 4-46
- *Creating libraries with the ARM RealView Librarian* on page 4-46.

4.11.1 Disassembling code

You can configure the CodeWarrior IDE to call the ARM `fromelf` utility to display information about an object file or library file. Typically, `fromelf` is used to generate an assembly listing for an object file but it can also provide other information about symbols, strings, and debug listings.

You can disassemble an object file built from:

- the currently open source file in the editor window
- selected source files in the project window.

You can also disassemble output that has been processed by the `fromelf` utility. See *Disassembling output ELF images* on page 4-46 for more information.

Note

To disassemble an image file built from your current project, you must configure your build target to call `fromelf` as a post-linker. See *Configuring target settings* on page 10-8 and *Configuring RealView fromelf* on page 10-76 for more information.

Disassembling from the editor window

To disassemble an object file built from the current editor window source file:

1. Ensure that the editor window (displaying the object file) is the currently selected window.
2. Select **Disassemble** from the **Project** menu. If the object file is up to date, the disassembled code is displayed in a new editor window. If the object file is not up to date, the CodeWarrior IDE compiles the source file first.
3. (Optional) Select **Save** from the **File** menu to save the disassembled source code.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Disassembling from the project window

To disassemble an object file or source file from the project window:

1. Ensure that the project window is the currently selected window.
2. Select one or more files to disassemble.

   Note

   You cannot select library files.

3. Either:
   - select **Disassemble** from the **Project** menu
   - right-click on the selected file and select **Disassemble** from the context menu.

   If the object file is up to date, the disassembled code is displayed in a new editor window. If the object file is not up to date, the CodeWarrior IDE compiles the source file first. Disassembled source for each selected file is displayed in its own editor window.
4. (Optional) Click on an editor window and select **Save** from the **File** menu to save the disassembled source code.

4.11.2 Converting output ELF images to other formats

You can configure the CodeWarrior IDE to call RealView `fromelf` to convert executable ELF output from the linker to a number of binary formats suitable for embedding in ROM, including:

- Plain binary
- Motorola 32-bit S-Record
- Intel 32-bit Hex
- Intel 32-bit Hex S-Record
- Byte Oriented (Verilog Memory Model) Hex
- ELF
- Structure Field Offsets.

See the *RealView Compilation Tools Linker and Utilities Guide* for more information on using `fromelf`, including information on output formats and splitting `fromelf` output for multiple memory banks.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To configure `fromelf` to process output images you must:

1. Configure the Target Settings configuration panel to call `fromelf` as a Post-linker. See *Configuring target settings* on page 10-8 for detailed instructions.
2. Configure the `fromelf` utility to generate the output you want. See *Configuring RealView fromelf* on page 10-76 for detailed instructions.
3. Select **Make** from the **Project** menu or click the **Make** button. The CodeWarrior IDE compiles and links your code to produce an executable ELF output file, and then calls `fromelf` to convert the output to the binary format of your choice.

   The converted output is saved in:

   `ProjectName\TargetName`

   together with the executable ELF output.

Note

The `fromelf` utility can only convert executable ELF to binary formats. It cannot convert object code or libraries.

4.11.3 Disassembling output ELF images

You can configure the CodeWarrior IDE to call `fromelf` to disassemble output ELF images:

1. Configure the Target Settings configuration panel to call `fromelf` as a Post-linker. See *Configuring target settings* on page 10-8 for detailed instructions.
2. Configure the `fromelf` utility to generate the output you want. See *Configuring output format* on page 10-77 for detailed instructions.

The CodeWarrior IDE compiles and links your code to produce an executable ELF output file, and then calls `fromelf` as a post-linker to disassemble the output.

See the *RealView Compilation Tools Linker and Utilities Guide* for detailed information on the command-line options to `fromelf`.

4.11.4 Creating libraries with the ARM RealView Librarian

To configure the CodeWarrior IDE to call the ARM RealView Librarian (`armar`) to output libraries in `ar` format, you must configure the Target Settings configuration panel to call the ARM RealView Librarian as the Linker. See *Configuring target settings* on page 10-8 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The ARM RealView Librarian combines object files from the compiler and assembler with any other object files in your build target, such as partially linked ELF output from a sub-project, into an object library. The output library is saved in the build target subdirectory of the project data directory:

*ProjectName**ProjectName*_Data*TargetName*

The easiest way to build library code is to use the ARM-supplied default project stationery to create a library project. See *Creating a project with a sub-project* on page 4-23 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4.12 Running batch files with the Batch File Runner

CodeWarrior for RVDS provides a Batch File Runner that you can use to run a DOS batch file from your project.

Note

Support for the Batch File Runner is deprecated and is to be removed in a future release.

To use the batch runner you must:

- Configure the file mappings for your build target to recognize `.bat` files.
- Configure the Batch File Runner as the post-linker in the Target Settings configuration panel.

  Note

  You can configure only one post-linker. This means that you cannot use the batch runner and `fromelf` in the same build target. However, you can run `fromelf` from the batch file.

- Add the batch file to the build target and then build the build target (using **Make** from the **Project** menu). The batch runner is run only after a successful link operation.

By default, the ARM project stationery supplied with RVDS is configured to recognize batch files. Therefore, if you use the ARM project stationery to create a new project, the `.bat` file mapping is already configured. However, if you have a legacy project, you must configure the file mappings for *every* build target to which you want to add batch files.

This section describes:

- *Configuring file mappings to recognize batch files*
- *Configuring the batch runner as the post-linker* on page 4-49
- *Adding batch files and making the build target* on page 4-50.

4.12.1 Configuring file mappings to recognize batch files

To configure the file mappings for a build target:

1. Select **Edit** → ***TargetName* Settings...** from the main menu to display the Target Settings window for the build target you want to configure.
2. Select **File Mappings** in the Target Settings Panels list to display the configuration panel.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click the entry for the `.c` mapping in the File Mappings list to select it.
4. Change the filename Extension, from `.c` to `.bat`.
5. Click the **Compiler** drop-down menu and select **None**.
6. Click **Add** to insert a new file mapping (Figure 4-19).

Figure 4-19 Adding batch files to mappings

7. Click **OK** to confirm your changes.

4.12.2 Configuring the batch runner as the post-linker

To run batch files from your project, you must configure the Batch File Runner to be run as a post-linker in the Target Settings configuration panel.

Note

Support for the Batch File Runner is deprecated and will be removed in a future release.

To configure the batch runner:

1. Select **Edit** → ***TargetName* Settings...** from the main menu to display the Target Settings window for the build target you want to configure.
2. Select **Target Settings** in the Target Settings Panels list to display the configuration panel.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click the **Post-linker** drop-down menu and select **Batch File Runner**. See Figure 4-20.

Figure 4-20 Configuring the Batch File Runner as post-linker

4. Click **OK** to confirm your changes.

4.12.3 Adding batch files and making the build target

You can add one or more batch files to any build target that is configured to accept files with a `.bat` extension. You can add any number of batch files to the build target, but the batch runner will only run the batch file listed *first* in the Link Order view of the project window.

Note

If you try to add batch files to build targets in the current project that are not configured to accept `.bat` file extensions, the CodeWarrior IDE displays a warning message and adds the batch files only to the correctly configured targets.

To add the batch files and make your build target:

1. Either:
   - select **Add Files…** from the **Project** menu
   - drag and drop the batch files onto the project window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

See the example in *Adding source files to the build targets* on page 4-9 for more information.

2. Touch your project files to ensure that the project is rebuilt.

Note

The Batch File Runner is executed only after a successful link step. If your project is up to date, the linker is not executed and the Batch File Runner is not run.

3. If you have added more than one batch file to the build target, select the **Link Order** tab to specify the batch file to be run. Drag the batch file you want to execute so that it is displayed before any other batch file (Figure 4-21).

Figure 4-21 Setting the link order

4. Select **Make** from the **Project** menu, or click the **Make** button to build your project. The first batch file in the link order list is executed after the linker has completed, regardless of the order of the batch files in the Files view.

   For example, in Figure 4-21, `build_2.bat` is executed after the link step is completed. If the link step fails, the batch file is not executed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 5
Working with the RealView Debugger

This chapter describes how to use the CodeWarrior IDE and the RealView Debugger to run and debug your code. It contains the following sections:

- *About working with the RealView Debugger* on page 5-2
- *Generating debug information* on page 5-3
- *Running and debugging your code* on page 5-7
- *Using the message window* on page 5-9.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5.1 About working with the RealView Debugger

You can call the RealView Debugger from the CodeWarrior IDE to either debug, or run images output from a `make` operation. This section gives an overview of how the RealView Debugger integrates with the CodeWarrior IDE.

For a quick tutorial on running the debugger from the Code Warrior IDE, see *Working with an existing ARM project* on page 4-5.

5.1.1 How the RealView Debugger work with the CodeWarrior IDE

The CodeWarrior IDE enables you to call the RealView Debugger with a number of optional arguments. When you select **Debug** or **Run** from the **Project** menu, the CodeWarrior IDE starts your debugger and instructs it to load the image file output from the current build target.

Note

The CodeWarrior IDE displays a **Debug** menu in the main menu bar. The **Debug** menu is not used by the CodeWarrior IDE for RVDS.

When the image is loaded into the debugger, all control passes to the debugger. You must use the debugger interface to perform operations such as stepping, inserting breakpoints, and examining memory.

For information on using the RealView Debugger, see:

- *RealView Debugger Essentials Guide*
- *RealView Debugger User Guide.*

Selecting debug options

There are a number of options to the ARM compiler that affect the quality of the debug view available to the debugger. You can set the debug configuration options in the compiler configuration panels. The debug options are used to create the Debug and Release build targets provided by the default project stationery. See *Predefined build targets* on page 4-4 for more information.

See *Configuring the RealView Compiler* on page 10-50 for more information on setting build options yourself.

See also *Working with multiple build targets and sub-projects* on page 3-40 for more information on using multiple build targets in your projects.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5.2 Generating debug information

You can configure the RealView tools to generate debug table information when your source code is compiled and assembled. There are two ways to enable debug table generation:

- Use the debug column in the project window to enable debug table generation for individual source files.
- Use the RealView Compiler and RealView Assembler configuration panels to configure the ARM tools to generate debug tables. If debugging is turned on in the configuration panels, debug tables are generated for all source files in the current build target.

  Note

  The debug settings specified in the configuration panels override the settings for individual source files. This means that you can generate debug table information for an individual source file if the configuration panel debug option is turned off, but you cannot turn off debug table generation for a specific source file if the configuration panel debug option is turned on.

This section describes:

- *Generating debug information for individual source files*
- *Generating debug information for all source files in a build target* on page 5-5.

5.2.1 Generating debug information for individual source files

You can enable debug table generation for one or more individual source files in the current build target provided that the ARM tools are not configured to generate debug information for the entire build target. If the ARM tools are configured to generate debug information, selecting or deselecting individual source files has no effect.

Note

You can also use the Project Inspector window to enable or disable debugging information. See *Examining and changing project information for a file* on page 3-36 for more information.

To generate debugging information for individual source files:

1. Select the build target for which you want to generate debug information. See *Setting the current build target* on page 3-42 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select the source files or groups for which you want to generate debug information:
   - Click in the Debug column next to a single source file to turn on debug table generation.
   - Alt+click in the Debug column next to a source file or group to enable debugging for all source files in the current build target.
   - Click in the Debug column next to a group to enable debugging for all source files in a group.

   Note

   To generate debug information for source files in a sub-target or sub-project you must open the build target or sub-project.

For selected files, the debug column displays a Debug Info marker (Figure 5-1) and marks the source files for recompilation.

Figure 5-1 Debug Info markers

No marker indicates that no debugging information is generated for source files in the group.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Select **Make** from the **Project** menu or click the **Make** button to build your project. The ARM tools generate debug information for selected source files only.

5.2.2 Generating debug information for all source files in a build target

When you click the **Debug** button, the debug information that is generated depends on whether or not any files are currently selected for debug table generation:

- If no files are individually selected for debug table generation, the CodeWarrior IDE selects all files in the current build target.
- If any file is selected for debug table generation, the CodeWarrior IDE does not change the debug selection settings.

Use the RealView Compiler and RealView Assembler configuration panels to turn on debug table generation. You can use these panels to generate debug table generation for each tool. If you want to generate debug information for all source files in your current build target, you must select the appropriate options for the RealView Compiler and RealView Assembler.

Note

The default ARM project stationery is configured to generate debug information for all source files in the Debug build target. See *Predefined build targets* on page 4-4 for more information.

Using the RealView Compiler configuration panels

The following example shows how to enable debug setting for the RealView Compiler. The procedure for the RealView Assembler is similar. To enable debug table generation for all C source files in a build target:

1. Open your project window and select the build target you want to configure. See *Selecting a build target* on page 4-36 for more information.
2. Click the **Target Settings** button to display the Target Settings window for the build target you want to configure. See *Displaying Target Settings panels* on page 10-4 for more information.
3. Click `RealView Compiler` in the Target Settings Panels list, and click the **Debug / Opt** tab to display the configuration panel (Figure 5-2 on page 5-6).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 5-2 RealView Compiler Debug / Opt configuration panel

4. Select **Enable debug table generation** to instruct the compiler to generate DWARF3 debug tables. See *Configuring debug and optimization* on page 10-59 for more information on the other options available on this panel. See *Configuring assembler options* on page 10-44 for detailed information on selecting Assembler debug options.

5. Click **Apply** to apply the settings.

6. Click **OK** to save the settings and close the dialog box.

When you make your project, the compiler generates debug tables for all C source files in the current build target, regardless of the debug settings of individual files in the target.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5.3 Running and debugging your code

This section describes how to run executable images from within the CodeWarrior IDE, and how to call the RealView Debugger to run or debug your code.

5.3.1 Running a project

To call the RealView Debugger to run an executable image from the CodeWarrior IDE:

1. Ensure that the project you want to run is the currently active window.
2. Select **Run** from the **Project** menu.

   The CodeWarrior IDE compiles and links the currently selected build target, if necessary, and creates an executable image file. It then executes the image file with the debugger selected in the RealView Runner configuration panel (see *Configuring the RealView Runner* on page 10-85).

   Note

   If the current build target is configured to produce non-executable output, such as a library or a partially linked object, the **Run** menu item is not available.

5.3.2 Debugging a project

To call the RealView Debugger to debug an executable image from the CodeWarrior IDE:

1. Ensure that the project you want to debug is the currently active window.
2. Ensure that you have configured your debugger and set the correct debug options for your build target. See:
   - *Configuring the RealView Debugger* on page 10-84 for more information.
   - *Configuring assembler and compiler language settings* on page 10-37 for information on how to enable debug table generation for the ARM assembler and the compiler.

   Note

   If your project is based on ARM-supplied stationery there are at least two separate build targets defined:
   - Debug
   - Release.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

If you are planning separate Debug and Release versions of your code, select **Project → Set Current Target… → Debug** to set the Debug build target. The Debug build target is configured to generate the most complete debug information at the expense of optimization.

---

3. Select **Debug** from the **Project** menu. The CodeWarrior IDE compiles and links your build target, if required, and calls the debugger you have specified in the RealView Debugger configuration panel.

See the *RealView Debugger Essentials Guide* for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5.4 Using the message window

This section describes the message window. The message window displays messages about events that have occurred when compiling, linking, or searching files. There are two basic types of message window:

- Errors & Warnings message window
- Notes message window.

These are described in:

- *Overview of the message window*
- *The Errors & Warnings message window* on page 5-10
- *Using the message window* on page 5-13.

5.4.1 Overview of the message window

The message window displays the following types of messages:

Errors Error messages are given by the compiler, assembler, linker, or `fromelf` in response to errors that prevent them from completing an operation. The final output from the tool is not created.

Warnings Warning messages are given by the compiler, assembler, or linker, in response to a problem, or potential problem from which the tool can recover. Problems that cause warning messages might result in problems in the final output file.

Notes These are informational messages that are given in response to an operation.

Note

Some Notes messages indicate a problem that is serious enough to stop output being produced.

The different message types are displayed in two variants of the message window:

Errors & Warnings message window

This message window displays error and warning messages from the compiler, assembler, linker, and post-linker (see *The Errors & Warnings message window* on page 5-10).

Notes message window

This message window displays all other types of message, including:

- results of a batch Find operation

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- messages displayed when the CodeWarrior IDE adds an access path to your project
- a message if you try to build an up-to-date project, if this option is selected in the Build Settings configuration panel in the IDE Preferences window. See *Configuring build settings* on page 9-6 for more information.

Error and warning messages are not displayed in the Notes message window.

Note

Not all interface elements appear in all message windows. For example, Notes message windows display the Source Code pane only if it is applicable to the specific message.

5.4.2 The Errors & Warnings message window

Figure 5-3 on page 5-11 shows an example of the Errors & Warnings message window.

The message window contains interface elements that you can use to perform common tasks such as:

- viewing error messages, warning messages, and other diagnostic messages
- navigating to locations in your source code that caused an error message or warning message.

This section describes the major interface components of the window.

Some user interface items in the message window are not described here. See *Overview of the editor window* on page 6-3 for more information on the:

- **Markers** drop-down menu
- **Document Settings** drop-down menu
- **Version Control** drop-down menu
- **Line Number** button
- File Path caption.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 5-3 Errors & Warnings message window

The major interface components of the Errors & Warnings message window are:

Errors button

The **Errors** button toggles the view of error messages on and off. See *Viewing error and warning messages* on page 5-13 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Warnings button

The **Warnings** button toggles the view of warning messages on and off. See *Viewing error and warning messages* on page 5-13 for more information.

Notes button

The **Notes** button toggles the view of informative notes on and off. See *Viewing error and warning messages* on page 5-13 for more information.

Project Information caption

The Project Information caption gives a short description of the view you are looking at in the message window. Your project name is displayed here.

Extra Information button

The **Extra Information** button expands a message to show information about the project, target, and file that caused a message.

Stepping buttons

The **Stepping** buttons enable you to step up or down through the messages in the window. See *Stepping through messages* on page 5-14 for more information.

Message List pane

The Message List pane displays your messages. See *Viewing error and warning messages* on page 5-13 for more information.

Source Code Disclosure triangle

The Source Code Disclosure triangle enables you to hide the Source Code pane of the message window.

Source Code pane

The Source Code pane of the message window enables you to view the source code at the location referred to by a message. See *Viewing error and warning messages* on page 5-13 for more information.

Pane resize bar

The Pane resize bar enables you to reallocate the amount of space in the message window given to the Source Code pane and Message List pane. Click and drag this bar up or down to change the amount of space on your computer screen that is allocated to both panes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5.4.3 Using the message window

The message window displays any error and warning messages given by the compiler, assembler, linker, and other tools when processing a menu command such as **Make**, **Bring Up To Date**, or **Check Syntax**. This section explains how to interpret, navigate, and use the information that appears in the message window. It describes:

- *Viewing error and warning messages*
- *Filtering error and warning messages*
- *Stepping through messages* on page 5-14
- *Correcting compilation errors and warnings* on page 5-14
- *Correcting link errors* on page 5-15
- *Printing the message window* on page 5-16
- *Saving the message window* on page 5-16.

Viewing error and warning messages

The message window is opened by the CodeWarrior IDE to display messages. To close the message window either:

- click its close box
- select **Close** from the **File** menu while the message window is the active window.

To reopen the message window, select **Errors & Warnings Window** from the **View** menu.

Note

This menu item is available only if a list of errors or warnings has already been generated as the result of a `make`, or compile operation.

Filtering error and warning messages

By default, both the **Errors** button and the **Warnings** button are selected when the CodeWarrior IDE displays an Errors & Warnings message window. You can specify which types of message you want to view:

- To view only error messages in the Message List pane, click the **Errors** button to deselect it and ensure that all other buttons are deselected.
- To view only warning messages in the Message List pane click the **Warning**s button to deselect it and ensure that all other buttons are deselected.
- To view only notes in the Message List pane click the **Notes** button to deselect it and ensure that all other buttons are deselected.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- To view a combination of message types, select or deselect buttons as required.

Stepping through messages

To move to a specific message in the list of messages displayed in the message window either:

- click the up and down **Stepping** buttons
- click the error message you want to select
- use the Up and Down Arrow keys.

As you move through the error and warning messages, the source code that caused the message is displayed in the Source Code pane.

Correcting compilation errors and warnings

To correct a compilation error or warning from the Source Code pane of the message window:

1. Ensure that the Source Code pane of the message window is visible. If it is not visible, click the Source Code Disclosure Triangle to display the pane (see Figure 5-3 on page 5-11).

2. Select a message in the Message List pane of the message window to view the statement that caused the message. The Source Code pane displays the source code that caused the message. A statement arrow points to the line of code that the compiler or assembler reports as an error (see Figure 5-4 on page 5-15).

   Note

   If you have corrected an error or modified the source code since the message list was generated, the CodeWarrior IDE might not be able to locate the correct position in the source code file. In this case, the CodeWarrior IDE displays an alert telling you that the position of the error or warning could not be found. You must recompile your project to update the list of errors and warnings in the message window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 5-4 Statement arrow pointing to an error

3. Either edit the source code directly in the Source Code pane or open the file in its own editor window.

   To open a source code file that corresponds to a given message either:

   - select the message in the Message List pane and press Enter
   - double-click the message in the Message List pane.

   Note

   You can use the **Header Files** drop-down menu, **Functions** drop-down menu, or the **Line Number** button to navigate the source code for a selected message. See *Overview of the editor window* on page 6-3 for more information.

Correcting link errors

There are many possible causes of link errors. Some of the most common are:

- You have misspelled the name of a library routine. This means that the routine that the linker is searching for does not exist.
- You are using inconsistent PCS options for the compiler and assembler. See *Configuring PCS options* on page 10-42 and *Configuring PCS and code generation options* on page 10-54 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- Your project is missing the necessary libraries. Check that your access paths include the directories where you store your libraries. See *Configuring access paths* on page 10-12 for more information.

  Check also that the setting for the **Search standard libraries** option is as you expect. If this option is selected, the RealView tools delivered with RVDS v3.1 use the `RVCT31LIB` environment variable to search for libraries. See *Configuring linker options* on page 10-69 for more information.

- You have not correctly linked the output from a dependent build target or sub-project. You must explicitly specify that output from a sub-project or sub-target is linked into your final output image. See *Working with multiple build targets and sub-projects* on page 3-40 for more information.

- You have not correctly set up your linker output options. See *Configuring the RealView Linker* on page 10-66 for more information.

Link errors are displayed in the message window in the same way as compilation errors.

Printing the message window

To print the message window:

1. Ensure that the message window you want to print is the active window.
2. Select **Print** from the **File** menu. The Print dialog box is displayed.
3. Select the print options you require and click **OK**. The messages are printed (marked by the date and time).

Saving the message window

To save a message window to a text file:

1. Ensure that the message window you want to save is the active window. You must click in the Error & Warnings message section of the window, not the Code section, in order to save the error and warning messages.
2. Select **Save A Copy As…** from the **File** menu. The CodeWarrior IDE displays a standard file dialog box.
3. Enter a name for the file and click **Save**. The CodeWarrior IDE saves the messages to a text file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copying the message window to the clipboard

To copy the contents of the message window to the Windows clipboard:

1. Ensure that the message window is the currently active window.
2. Select **Copy** from the **Edit** menu. The messages are copied to the clipboard. You can paste the clipboard contents into a text editor or other application.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 6
Editing Source Code

This chapter describes how to use the CodeWarrior IDE text editor to edit your source code. It contains the following sections:

- *About editing source code* on page 6-2
- *Overview of the editor window* on page 6-3
- *Configuring the editor window* on page 6-7
- *Editing text* on page 6-10
- *Navigating text* on page 6-19.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.1 About editing source code

The CodeWarrior IDE editor is a fully-featured text editor designed for programmers. It includes:

- drop-down menus on every editor window for opening header files and quickly navigating among functions
- integrated VCS menus that enable you to work with your version control system from within the CodeWarrior IDE

  This menu is only available if a version control system plug-in module has been installed. All current VCS modules which can be used with the CodeWarrior IDE are available from the Metrowerks web site.
- syntax highlighting that formats source code for easy identification of comments and keywords in your source files.

This chapter describes the basics of how to use the CodeWarrior IDE editor. See also:

- Chapter 2 *Working with Files* for information on basic file operations such as opening, saving, and comparing source files
- Chapter 7 *Searching and Replacing Text* for information on searching and replacing text in source files
- Chapter 8 *Working with the Browser* for information on using the CodeWarrior IDE source code browser
- Chapter 9 *Configuring IDE Options* for information on customizing the way the CodeWarrior IDE editor works.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.2 Overview of the editor window

The editor window provides drop-down menus and other controls that enable you to perform basic editing operations. Figure 6-1 shows an example of the CodeWarrior IDE editor window.

Figure 6-1 The editor window

The major interface components of the editor window are:

Header Files drop-down menu

You can use the **Header Files** drop-down menu (see Figure 6-2 on page 6-4) to:

- open header files referenced by the current file
- use the **Touch** and **Untouch** commands.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 6-2 Header Files drop-down menu

See *Touching and untouching files* on page 3-34 for more information.

Functions drop-down menu

You can use the **Functions** drop-down menu (Figure 6-3) to jump to a specific function in another text file within your source code.

Figure 6-3 Functions drop-down menu

See *Using the Functions drop-down menu* on page 6-19 for more information.

Markers drop-down menu

You can use the **Markers** drop-down menu (Figure 6-4) to add and remove markers in your text files.

You can use markers for:

- quick access to a line of text
- remembering where you left off
- other identification purposes.

Figure 6-4 Markers drop-down menu

See *Using markers* on page 6-21 for more information.

Document Settings drop-down menu

You can use the **Document Settings** drop-down menu (see Figure 6-5 on page 6-5), to turn color syntax highlighting on or off for the current file, and to set the method for saving the file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 6-5 Document Settings drop-down menu

See *Text Colors* on page 9-32 for details of how to modify syntax coloring.

Version Control drop-down menu

The **Version Control** drop-down menu indicates the read/write revision control database status of the current file.

Note

This menu is only displayed if a version control system plug-in module has been installed. All current VCS modules which can be used with the CodeWarrior IDE are available from the Metrowerks web site.

Browser context menu

The **Browser** context menu is a context-sensitive menu that provides quick access to browser information. To access it, right-click on any symbol name in the window.

See *Using the Browser context menu from an editor window* on page 8-22 for information on using this menu.

Note

The **Browser** context menu displays a **Set Breakpoint** menu item in source code windows. This command is not implemented in the ARM® version of the CodeWarrior IDE.

File Path caption

The CodeWarrior IDE displays the directory path of the current file in the File Path caption.

Dirty File marker

The Dirty File marker shows if the file displayed in a window has been modified after it was last saved or opened.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The states of the Dirty File marker are:

Unchanged file

Modified and unsaved file (*dirty*)

Pane splitter controls

Pane splitter controls split the editor windows into panes so you can view different portions of a file in the same window. Use these controls to adjust the sizes of the panes after you have created them.

See *Splitting the window into panes* on page 6-8 for more information on pane splitter controls (Figure 6-7 on page 6-8 shows an editor window with multiple panes).

Line Number button

The **Line Number** button displays the number of the line that contains the text insertion point. You can click on this button to go to another line in the file.

See *Going to a specific line* on page 6-24 for more information on setting the text insertion point on another line.

Toolbar Disclosure button

The **Toolbar Disclosure** button hides or displays the editor window toolbar along the top of the window. If the toolbar is hidden, a row of controls is displayed at the top of the editor window.

See *Displaying window controls* on page 6-7 for more information on using the **Toolbar Disclosure** button.

Text editing area

The text editing area of the editor window is where you enter and edit text. See *Editing text* on page 6-10.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.3 Configuring the editor window

The editor enables you to customize your view of the file with which you are working. This section describes options available in the editor window:

- *Setting text size and font*
- *Displaying window controls*
- *Splitting the window into panes* on page 6-8
- *Saving editor window settings* on page 6-9.

See *Customizing toolbars* on page 9-48 for more information on configuring the editor window toolbar.

6.3.1 Setting text size and font

Use the Font & Tabs configuration panel (in the IDE Preferences window) to set the size or font used to display text in an editor window. See *Font & Tabs* on page 9-29 for more information.

6.3.2 Displaying window controls

The toolbar comprises the row of drop-down menus and controls that appears along the top of the editor window.

To hide the toolbar, click the **Toolbar Disclosure** button, shown in Figure 6-1 on page 6-3. The CodeWarrior IDE hides the toolbar, and displays the default toolbar drop-down menu controls along the bottom of the editor window (Figure 6-6).

Figure 6-6 Default toolbar at bottom of editor window

The File Path caption is no longer visible when the toolbar is hidden.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

If you have customized the editor window toolbar, your custom toolbar items are not displayed at the top of the window. When you display the toolbar again, its custom configuration is restored.

To re-show the toolbar if it is hidden, click the **Toolbar Disclosure** button. The toolbar is displayed along the top of the editor window.

See *Customizing toolbars* on page 9-48 for general information on toolbars, including toolbar customization.

You can select a default setting to display or hide the toolbar in editor windows. See *Showing and hiding a toolbar* on page 9-49 for more information.

6.3.3 Splitting the window into panes

You can split the editor window into panes to view different parts of a file in the same window. Figure 6-7 shows an example.

Figure 6-7 Multiple panes in an editor window

See:

- *Creating a new pane* on page 6-9
- *Resizing a pane* on page 6-9
- *Removing a pane* on page 6-9.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Creating a new pane

You can click and drag a pane splitter control to create a new pane in an editor window. Pane splitter controls are on each scroll bar of a pane in the editor window (see Figure 6-1 on page 6-3). To use a pane splitter control:

1. Drag the pane splitter control toward the desired location of the new pane. As you drag the control, a gray focus bar tracks your progress and indicates where the new pane will go.
2. Release the mouse button to create a new pane.

Alternatively, you can double-click the pane splitter control to split a pane into two equal parts.

Resizing a pane

To resize a pane:

1. Click and drag the pane resize boxes to change the sizes of the panes in an editor window. As you drag a resize box, a gray focus line indicates your progress.
2. Release the mouse button to redraw the pane in its new position.

Removing a pane

To remove a pane from an editor window:

1. Click and drag a resize box to any edge of the window. As you drag the resize box, a gray focus box indicates your progress. If you drag the box close to the edge of the window, the gray lines are no longer displayed.
2. Release the mouse button when the gray lines are no longer displayed. The editor removes one of the panes from the window.

Alternatively, you can double-click on a resize bar to remove a split.

6.3.4 Saving editor window settings

The current settings of an editor window are automatically saved when:

- you close the window
- the toolbar is hidden or displayed.

The settings that are saved include the size and location of the window, and the display of the toolbar. When you re-open an editor window, the CodeWarrior IDE uses the saved window settings.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.4 Editing text

The CodeWarrior IDE provides many methods for editing source files. These are described in:

- *Basic editor window navigation*
- *Basic text editing* on page 6-11
- *Selecting text* on page 6-12
- *Moving text with drag and drop* on page 6-13
- *Balancing punctuation* on page 6-14
- *Formatting code* on page 6-14
- *Completing code* on page 6-15
- *Shifting text left and right* on page 6-16
- *Undoing changes* on page 6-17
- *Controlling color* on page 6-18.

6.4.1 Basic editor window navigation

This section describes basic text navigation techniques and shortcuts you can use in text editor windows.

Scrollbar navigation

Use the scrollbars to adjust the field of view in an editor window in the CodeWarrior IDE.

Keyboard navigation

Table 6-1 shows the keystrokes you can use to move the insertion point in a file.

Table 6-1 Text navigation with the keyboard

To move insertion point to…	Keystroke
Previous word	Ctrl+Left Arrow
Next word	Ctrl+Right Arrow
Beginning of line	Home
End of line	End
Beginning of file	Ctrl+Home
End of file	Ctrl+End

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table 6-2 shows the keystrokes you can use to scroll to different locations in a file, without moving the insertion point.

Table 6-2 Scroll with the keyboard

To scroll to…	Keystroke
Previous page	Page Up
Next page	Page Down
Previous line	Ctrl+Up Arrow
Next line	Ctrl+Down Arrow

6.4.2 Basic text editing

The CodeWarrior IDE supports the standard Windows editing operations provided by most Windows text editors:

Adding text Click once in the text editing area of the window to set the new location of the text insertion point and begin typing to enter text.

Also, see *Balancing punctuation* on page 6-14 for information on helping you to ensure consistency in the text that you enter.

Deleting text

Press the Backspace key to delete text that is behind the text insertion point or the Delete key to delete text that is in front of the text insertion point.

You can also select text and press the Backspace or Delete key to delete the selection. See *Selecting text* on page 6-12 for details on how to select text.

Cut, copy, paste

You can use the standard Windows editing commands to remove text, or to copy and paste in a window, between windows, or between applications. See *Edit menu* on page D-5 for information on these commands.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.4.3 Selecting text

This section describes how to select text in editor windows:

- *Selecting text using keystroke shortcuts*
- *Selecting text using the mouse* on page 6-13.

Selecting text using keystroke shortcuts

To select text using keystroke shortcuts, hold down the Shift key while pressing a text navigation key sequence.

Table 6-3 shows the keystrokes for selecting text, starting at the current insertion point.

Table 6-3 Text selection with the keyboard

To select...	Keystroke
Previous word	Shift+Ctrl+Left Arrow
Next word	Shift+Ctrl+Right Arrow
Beginning of line	Shift+Home
End of line	Shift+End
Beginning of page	Shift+Page Up
End of page	Shift+Page Down
Beginning of file	Shift+Ctrl+Home
End of file	Shift+Ctrl+End

To select blocks of code quickly, use the **Balance** command. See *Balancing punctuation* on page 6-14 for more information.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Selecting text using the mouse

Table 6-4 gives a summary of how to select text with the mouse.

Table 6-4 Selecting text with the mouse

To select a…	Do this…
Single word	Double-click on the word.
Single line	Either: • triple-click anywhere in the line • move the mouse pointer to the left edge of the editor window so the mouse pointer points right, and press the mouse button. This selection method is available when the **Left margin click selects line** option is on in the Editor Settings configuration panel.
Range of text	Use any of the following methods: • Click and drag the mouse in a portion of your window where there is text. • Set your text insertion point to mark the beginning of your selection, and press the Shift key while clicking the place in your text where you want the selection to end. • Move the mouse pointer to the left edge of the editor window so the mouse pointer points right, and click and drag the mouse pointer to select lines of text. This selection method is available when the **Left margin click selects line** option is on in the Editor Settings configuration panel.
Function	Press the Shift key while selecting a function in the **Functions** drop-down menu to display and highlight an entire function in the editor window. This is particularly useful for copy and paste operations, and for using drag and drop to move code around in your file.

6.4.4 Moving text with drag and drop

Use the drag and drop features of the editor if you have text in your editor window that you want to move to a new location. To use drag and drop editing, you must enable this feature in the CodeWarrior IDE Preference panels. See *Editor settings* on page 9-24 for information on how to turn this feature on or off.

The CodeWarrior IDE editor accepts drag-and-drop text items from other applications that support drag-and-drop editing.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Drag and drop text in any of the following ways:

- Select and drag text from an editor window to any destination that can accept a drop. You can drag and drop text to a new location in the current editor window, or to another open editor window.
- Drag selected text into an editor window from other applications that support drag and drop.
- Drag and drop an icon of a text file directly into the editor window.

6.4.5 Balancing punctuation

The CodeWarrior IDE provides manual balancing and automatic balancing to ensure that every parenthesis (`()`), bracket (`[]`), and brace (`{}`) in your code has a counterpart, where applicable.

Using manual balancing

To check for balanced parentheses, brackets, or braces, either:

- Place the insertion point in the text you want to check and select **Balance** from the **Edit** menu.
- Double-click on a parenthesis, bracket, or brace character that you want to check for a matching character.

The CodeWarrior IDE editor searches from the text insertion point until it finds a close parenthesis, bracket, or brace, and then searches in the opposite direction until it finds a matching open parenthesis, bracket, or brace. When it finds the match, it selects the text between them. If the insertion point is not enclosed, or if the punctuation is unbalanced, the CodeWarrior IDE emits a warning beep.

Note

You can use the **Balance** command to select blocks of code quickly.

6.4.6 Formatting code

The Code Warrior IDE provides options for automatically formatting code as it is entered in the editor window. To use code formatting, you must enable this feature in the CodeWarrior IDE Preference panels.

You can set code formatting for use with either C/C++ code or Java code and there are a number of code formatting options you can choose from depending on the coding standards you are using.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Code formatting works automatically as you enter code. For example, if you have enabled the **Format braces** option, then when you enter a single brace in the editor window an additional matching brace is automatically added and the insertion point is placed between the braces.

See *Code formatting* on page 9-22 for information on how to enable this feature, and a complete list of the code formatting options available.

6.4.7 Completing code

The Code Warrior IDE provides code completion options for automatically suggesting ways to complete source code as it is entered in the editor window. By using code completion you avoid referring to other files to remember available symbols, thereby reducing the potential for introducing errors in your code. To use code completion, you must enable this feature in the CodeWarrior IDE Preference panels.

Note

You can use C/C++ code completion more effectively when **Language Parser** is selected for the **Generate Browser Data From** option in the **Build Extras** tab. The CodeWarrior IDE is then able to use the database of symbols defined in the browser to provide code completion suggestions. See *Configuring build extras* on page 10-21.

To activate code completion:

1. Begin typing in an editor window or place an insertion point at the end of the source code that you want to complete.
2. Select **Complete Code** from the **Edit** menu, or type Alt+. (Alt+period).

   The Code Completion window is displayed beneath the source code containing suggestions for completing the code. See Figure 6-8.

Figure 6-8 Code Completion window

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. The window lists available variables and methods or functions along with their corresponding return types or parameters. The list of items shown in the window changes based on the context of the insertion point in the active editor window.

   Use the information displayed in the Code Completion window to help you complete the code at the insertion point.

   To select the next code completion item, type Alt+/. To select the previous code completion item, type Alt+Shift+/.

You can use the following commands to cycle through code completion suggestions in the editor window:

- To display the next code completion item select **Get Next Completion** from the **Edit** menu (or type Alt+/).
- To display the previous code completion item select **Get Previous Completion** from the **Edit** menu (or type Alt+Shift+/).

The items will be displayed in the editor window alongside the cursor position as you cycle-through the suggestions.

See *Code completion* on page 9-21 for information on how to enable and configure this feature.

6.4.8 Shifting text left and right

You can format your source code by shifting blocks of text left and right. This enables you to indent large blocks of text easily.

To shift blocks of text left and right:

1. Select a block of text (see *Selecting text* on page 6-12).
2. Select **Shift Right** or **Shift Left** from the **Edit** menu.

   The CodeWarrior IDE editor shifts the selected text one Tab stop to the right or left by inserting or deleting a Tab character at the beginning of every line in the selection.

See *Font & Tabs* on page 9-29 for information on configuring the number of spaces defined for a Tab character.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.4.9 Undoing changes

The CodeWarrior IDE editor provides several methods to undo mistakes as you edit a file. The available methods are:

- *Undoing the last edit*
- *Undoing and redoing multiple edits*
- *Reverting to the last saved version of a file* on page 6-18.

Undoing the last edit

The **Undo** command reverses the effect of your last action. The name of the undo command on the **Edit** menu changes depending on your last action. For example, if your most recent action was to type some text, the command changes to **Undo Typing**. In this case, you can select **Undo Typing** to remove the text you typed.

Undoing and redoing multiple edits

You can use multiple undo and redo commands when the **Use Multiple Undo** option is selected in the Editor Settings IDE Preferences configuration panel. See *Configuring other settings* on page 9-11 for information on how to enable the **Use multiple undo** option.

When multiple undo is enabled, you can select **Undo** or **Redo** from the **Edit** menu multiple times to undo and redo your previous edits.

For example, if you cut a word, paste it, then type some text, you can reverse all three actions by choosing **Undo** three times. The first undo removes the text you typed, the second unpastes the text you pasted, and the third reinstates the text you cut to restore the text to its original condition.

You can redo the edits by selecting **Redo** three times.

Note

The keyboard shortcut for the **Redo** command changes when the **Use multiple undo** option is turned off.

Undo actions are saved in a stack. Each undo action adds an item to the stack, and each redo repositions a pointer to the next undo action. If you perform several undo and redo actions you lose actions off the stack. For example, if there are five undo actions on the stack (`ABCDE`), and you redo two of them, the stack appears to the undo pointer as `ABC`. When you perform a new action (`ABCF`), the undo events (`DE`) are no longer available.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Reverting to the last saved version of a file

You can discard all changes you have made since the last time you saved your file. Select **Revert** from the **File** menu to return a file to its last-saved version. See *Reverting to the most recently saved version of a file* on page 2-20 for more information.

6.4.10 Controlling color

You can use color to highlight many elements in your source code, such as comments, keywords, and quoted character strings. You can also highlight custom keywords that are in a list of words you designate. See *Text Colors* on page 9-32 for information on configuring color syntax options.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.5 Navigating text

The CodeWarrior IDE editor provides several methods for navigating a file that you are editing.

This section describes the following methods:

- *Finding a function*
- *Finding symbol definitions* on page 6-20
- *Using markers* on page 6-21
- *Using bookmarks* on page 6-23
- *Going to a specific line* on page 6-24
- *Using Go Back and Go Forward* on page 6-24.
- *Opening a related header file* on page 6-25.

See also Chapter 8 *Working with the Browser* for details on the methods provided by the integrated code browser for navigating through code.

Note

You can customize key bindings for the CodeWarrior IDE editor. See *Customizing key bindings* on page 9-42 for more information on how to change key bindings that move the text insertion point around in a file.

6.5.1 Finding a function

You can use the **Functions** drop-down menu to find a specific function within the source file currently displayed in the editor window.

Note

- If the drop-down menu is empty, the current editor file is not a source code file.
- You cannot use the **Functions** drop-down menu to navigate through ARM assembly language code.

Using the Functions drop-down menu

To jump to a specific function in the current source file:

1. Ensure that the editor window that contains the function is the currently active window.
2. Click on the **Functions** drop-down menu (see Figure 6-9 on page 6-20). The drop-down menu lists the functions in your source file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

By default, the menu lists the functions in the order in which they appear in the file. You can list functions in alphabetical order by pressing the Ctrl key before you click on the **Functions** drop-down menu.

Note

You can change the default order of the functions in the **Functions** drop-down menu if the **Sort Function Popup** option is selected in the Editor Settings IDE Preferences configuration panel. See *Editor settings* on page 9-24 for more information.

If the function name in the replace list has a bullet next to it, it means that the text insertion point is currently located within the definition of that function (Figure 6-9).

Figure 6-9 Functions in the Functions drop-down menu

6.5.2 Finding symbol definitions

You can find symbol definitions in any source file within your current project.

To look up the definition of a symbol:

1. Select the symbol name in your source code.
2. Select **Find Definition** from the **Search** menu. Alternatively, you can press the Alt key and double-click on the symbol name. The CodeWarrior IDE searches all the files in your project for the definition of the symbol.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

If the CodeWarrior IDE finds one or more matches in your project, it opens a window and displays each of the matches for you to examine. If the browser is enabled for your project, the CodeWarrior IDE displays the browser Symbol window (see *Finding overrides and multiple implementations of a function* on page 8-26). Otherwise, the CodeWarrior IDE displays a message window (see *Using the message window* on page 5-13).

6.5.3 Using markers

You can add or remove a marker in any of your text files using the facilities built into the CodeWarrior IDE editor. Markers set places in your file that you can name. Markers are saved with the file. You can then jump to the marker quickly. You can also use markers for leaving notes to yourself about work in progress on your code. See:

- *Adding a marker*
- *Removing a marker* on page 6-22
- *Jumping to a marker* on page 6-22.

You can also use bookmarks to set places on lines in your file which are not named. See *Using bookmarks* on page 6-23 for more information.

Adding a marker

To add a marker:

1. Move the text insertion point to the location in the text you want to mark.
2. Select **Add marker...** from the **Markers** drop-down menu. The CodeWarrior IDE displays an Add Marker dialog box (Figure 6-10).

Figure 6-10 Add Marker dialog box

3. Enter text in the dialog box to mark the location in the file with a note, comment, function name, or other text that would be helpful to you. Markers do not have to be unique but doing this will help you to find them quickly.
4. Click **Add**. Your marker is visible in the **Markers** drop-down menu (see Figure 6-11 on page 6-22).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 6-11 Example text file with a marker added

Markers are shown in this list the order they were added, not their order in the file.

Note

If you select some text in a source file, then select **Add marker...**, the selected text appears as the new marker name in the Add Marker dialog box. This is useful for quickly adding specific functions or lines as markers.

Removing a marker

To remove a marker:

1. Click the **Markers** drop-down menu and select the **Remove markers** command. The CodeWarrior IDE displays the Remove Markers dialog box (Figure 6-12).

Figure 6-12 Remove Markers dialog box

2. Select the marker you want to delete and click **Remove** to remove it permanently from the marker list.

3. Click **Done** to close the Remove Markers dialog box.

Jumping to a marker

To jump to a marker:

1. Click the **Markers** drop-down menu.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select the name of the marker from the list shown on the menu to set the text insertion point at the marker location.

6.5.4 Using bookmarks

You can add or remove bookmarks in any of your text files using the facilities built into the CodeWarrior IDE editor. Bookmarks are similar to markers in that they set places in the code you can jump to. However, they differ from markers:

- Bookmarks are placed on a line so you can only have one bookmark on a line at one time.
- Bookmarks cannot be given a name.

Adding a bookmark

To add a bookmark:

1. Move the text insertion point to the location in the text you want to mark.
2. Type Ctrl+F2.

   A bookmark (shown as a small blue icon) is placed at the beginning of the line containing the text insertion point (Figure 6-13).

Figure 6-13 Editor window containing a bookmark

To remove a bookmark

To remove a bookmark, move the text insertion point to a position in the line containing the bookmark and type Ctrl+F2. The bookmark is removed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To jump to a bookmark

To jump to the next bookmark in a file, type F2. To jump to the previous bookmark in a file, type Shift+F2. The bookmark search will wrap to the beginning of the file when it reaches the end of the file.

6.5.5 Going to a specific line

You can go to a specific line in an editor window if you know its number. Lines are numbered consecutively, with the first line designated as line 1. To go to a particular line:

1. Select **Go to Line...** from the **Search** menu, or click **Line Number** on the editor window, to open the Line Number dialog box (Figure 6-14).

Figure 6-14 Line Number dialog box

2. Enter the number of the line you want to jump to.
3. Click **OK**.

6.5.6 Using Go Back and Go Forward

The **Go Back** and **Go Forward** commands are only available from the **Search** menu when you use the browser. See Chapter 8 *Working with the Browser* for more information on using the browser.

If you already have the browser enabled, see *Using Go Back and Go Forward* on page 8-21 for details on using these commands.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6.5.7 Opening a related header file

The CodeWarrior IDE enables you to open header files for the active editor window. You can open a header file in two ways:

- *Using the Header Files drop-down menu*
- *Using a keyboard shortcut* on page 6-26.

Using the Header Files drop-down menu

You can use the **Header Files** drop-down menu to open header files referenced by the file in the current editor window.

Note

You can also open the **Header Files** drop-down menu from the project window. See *Creating and opening files* on page 2-3 for more information.

You can also use the **Touch** and **Untouch** commands from the **Header Files** drop-down menu. See *Touching and untouching files* on page 3-34 for more details.

Some files cannot be opened in the editor window, such as libraries.

To use the **Header Files** drop-down menu to open a header file:

1. Ensure that the project in which the source file is included is open.

   If the project file is not open, the **Header Files** drop-down menu does not display a list.

2. In the editor window for the source file, click the **Header Files** drop-down menu to display the drop-down menu (Figure 6-15).

Figure 6-15 Header Files drop-down menu

3. Select the header file you want to open from the drop-down menu. The header file is opened in a new editor window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Using a keyboard shortcut

You can open a header file using a keyboard shortcut:

1. Select the filename of the header file in the active editor window.
2. Type Ctrl+D. The header file is opened in a new editor window.

See *Creating and opening files* on page 2-3 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 7
Searching and Replacing Text

This chapter describes how to use the CodeWarrior IDE search and replace functions. It contains the following sections:

- *About finding and replacing text* on page 7-2
- *Finding and replacing text in a single file* on page 7-3
- *Finding and replacing text in multiple files* on page 7-8
- *Using grep-style regular expressions* on page 7-17.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7.1 About finding and replacing text

The search and replace facilities in the CodeWarrior IDE enable you to search and replace from either the current editor window, or from the Find dialog box. You can also search using regular expressions, similar to the UNIX `grep` command.

The Find dialog box provides comprehensive search and replace facilities. You can use the dialog box to perform find and replace operations for:

- text in a single file
- text in multiple files in your project
- text in arbitrary files that are not part of your current project.

You can define a scope for the search to search through:

- all text (code and comments)
- code only
- comments only.

You can use text strings, text substrings, and pattern matching in find and replace operations. In addition, you can use the batch search option to display the results of a find operation in a text window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7.2 Finding and replacing text in a single file

This section describes how to use the CodeWarrior IDE search functions to locate specific text you want to replace in the active editor window. See:

- *Searching for selected text*
- *Finding and replacing text with the Find and Replace dialog box* on page 7-4.

7.2.1 Searching for selected text

The CodeWarrior IDE provides two ways of searching for text selected in the editor window, without opening the Find dialog box.

However, when you do the search, the CodeWarrior IDE uses the option settings that you last chose in the Find dialog box. To change the option settings, therefore, you must open the Find dialog box.

Finding text in the active editor window

To find text in the active window:

1. Select an instance of the text you want to find.
2. Select **Find Selection** from the **Search** menu. The CodeWarrior IDE searches for the next occurrence of your text string in the current file only.
3. Either:
   - select **Find Next** from the **Search** menu, or press F3, to search for the next occurrence of the text string
   - press Shift+F3 to search toward the beginning of the file for the previous occurrence of the text string.

Finding text in another window

This method is useful if you are working with a file in an editor window and you want to find occurrences of a text string in another file. To find text in another editor window:

1. Select an instance of the text you want to find.
2. Select **Enter Find String** from the **Search** menu. The editor enters the selected text into the Find text field of the Find dialog box.
3. Click on the editor window that you want to search to make it active.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Either:
   - select **Find Next** from the **Search** menu, or press F3, to search for the next occurrence of the text string.
   - press Shift+F3 to search toward the beginning of the file for the previous occurrence of the text string.

7.2.2 Finding and replacing text with the Find and Replace dialog box

To find and replace text with the Find and Replace dialog box:

1. Select **Replace…** from the **Search** menu. The Find and Replace dialog box is displayed (Figure 7-1).

Figure 7-1 Find and Replace dialog box

2. Type a text string into the **Find** text field, or select a string from the **Recent Strings** drop-down menu (in the **Find** text field).

   The **Recent Strings** drop-down menu contains strings that have previously been used for searches. Select an item in one of these drop-down menus to place it into the corresponding text box.

   You can use Cut, Copy, and Paste commands to edit text in the Find text field.

   Note

   See *Searching for special characters* on page 7-6 for information on how to search for a Return or Tab character.

3. Type a text string into the **Replace** text field, or select a string from the **Recent Strings** drop-down menu (in the **Replace** text field), if you want to replace the found string.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select the search options you want:

Match whole word

Select this option to find only complete words (delimited by punctuation or white-space characters) that match the search string. When this option is not selected, the CodeWarrior IDE finds occurrences of the search string embedded within larger words.

Case sensitive

Select this option to treat uppercase and lowercase text in the search string as distinct. When this option is not selected, uppercase and lowercase text are identical.

Regular expression

Select this option to instruct the CodeWarrior IDE to interpret the text in the Find text field as a regular expression. See *Using regular expressions* on page 7-18 for more information.

Stop at end of file

Select this option if you want the search to stop when it gets to the end of the file. When this option is not selected, the CodeWarrior IDE searches from the current insertion point to the end of the file, and continues to the insertion point.

Search Up

Select this option if you want to search the file in an upward direction from the insertion point. If this option is not selected the file is searched in a downward direction.

Search selection only

You can select a block of text within the file which you want to search. Select this option to instruct the CodeWarrior IDE to search only the currently selected text.

Scope Select an option from this group box to define the scope of the search. This can be one of:

- all text
- code only
- comments only.

5. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace operation:

- Click **Find** to search forward from the text insertion point in the active editor window. The CodeWarrior IDE finds the first instance of the string. Press F3 to search for subsequent instances.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- Select **Replace** from the **Search** menu to search forward from the text insertion point in the active editor window and find the first instance of the string. The CodeWarrior IDE replaces found text with the text in the Replace text field. To replace subsequent instances, select **Replace and Find Next** from the **Search** menu. Alternatively, use the keyboard shortcut Ctrl+L. Press Shift if you want to replace and find backwards.
- Select **Replace & Find Next** from the **Search** menu to replace found text and find the next occurrence of the find string. Press Shift to replace and search backwards.
- Select **Replace All** from the **Search** menu to replace all occurrences of the find string in a single operation. A confirmatory dialog box is displayed showing the number of replacements made.

  Caution

  Undo is not available for the **Replace All** command. It is recommended that you save your source file before using **Replace All**, so that you can use the **Revert** command if you want to discard the changes.

- Click **Cancel** to close the dialog box.

Searching for special characters

To enter a Tab or Return character in the Find or Replace fields, you must either:

- copy and paste your selected text with the Tab or Return characters into the Find text field
- enable the Regular expression option, and enter `\t` for Tab or `\r` for Return into the field.

Using regular expressions changes the way in which the CodeWarrior IDE locates a string match. See *Using grep-style regular expressions* on page 7-17 for more details.

The Search Results window

If you have opened the Find dialog box, click **Find All** to collect all successful matches of your search text in one window for easy reference. The results are displayed in a Search Results message window (see Figure 7-2 on page 7-7).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 7-2 Search Results window

Use the Search Results message window to navigate through the search results. For example, click on an entry in the list view to display the match in the source view pane, or double-click on an entry to display it in an editor window. See *Using the message window* on page 5-9 for more information on the features of the message window.

The Find dialog box

The Find dialog box is a cut-down version of the Find and Replace dialog box. You can use the Find dialog box to search for text patterns in the active editor window. When you find the text you are searching for, you can change it or look for the next occurrence.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7.3 Finding and replacing text in multiple files

The CodeWarrior IDE Find in Files dialog box enables you to search for text patterns in:

- directories
- projects
- symbolics
- files.

You can perform these different powerful searches by choosing the **In Folders**, **In Projects**, **In Symbolics**, or **In Files** view tabs and setting their specific search options.

In addition, you can save sets of search files for future use.

This section describes:

- *Using multifile search*
- *Using file sets* on page 7-15.

Note

You can also select the browser **Go Back** and **Go Forward** commands from the **Search** menu to access information and search through multiple files. See *Using Go Back and Go Forward* on page 8-21 for more information.

7.3.1 Using multifile search

This section describes how to perform multifile searches. See *Using file sets* on page 7-15 for more information on configuring multifile searches. To search for text in multiple files:

1. Select **Find in Files…** from the **Search** menu. The Find in Files dialog box is displayed (see Figure 7-3 on page 7-9).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 7-3 Find in Files dialog box

2. Select the search options you want:

   Match whole word

   Select this option to find only complete words (delimited by punctuation or white-space characters) that match the search string. When this option is not selected, the CodeWarrior IDE finds occurrences of the search string embedded within larger words.

   Case sensitive

   Select this option to treat uppercase and lowercase text in the search string as distinct. When this option is not selected, uppercase and lowercase text are identical.

   Regular expression

   Select this option to instruct the CodeWarrior IDE to interpret the text in the Find text field as a regular expression. See *Using regular expressions* on page 7-18 for more information.

3. Select the appropriate tab for your search:
   - **In Folders** (see *Searching through multiple files by directories* on page 7-10).
   - **In Projects** (see *Searching through multiple files by projects* on page 7-11).
   - **In Symbolics** (see *Searching through multiple files by symbols* on page 7-13).
   - **In Files** (see *Searching through multiple files by file sets* on page 7-14).

4. Type the search text into the Find text box.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5. Type the replacement text into the Replace with box if required.

6. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace, operation:

   - Click **Find All** to collect all successful matches of your search text in one window for easy reference. See *The Search Results window* on page 7-6 for information on using **Find All**.
   - Click **Replace** or select **Replace** from the **Search** menu to replace found text with the text in the Replace text field.
   - Click **Replace All** or select **Replace All** from the **Search** menu to replace all occurrences of the find string.

     Caution

     Undo is not available for the **Replace All** command. It is recommended that you save your source file before using **Replace All**, so that you can use the **Revert** command if you want to discard the changes.

Searching through multiple files by directories

To search multiple files by directories:

1. Click the **In Folders** tab in the Find in Files dialog box (Figure 7-4).

Figure 7-4 In Folders tab, Find in Files dialog box

2. Use the **Search in** drop-down menu to specify the location you want to search. Type in a location or click **Browse...** to find the required location.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Select **Search subfolders** if you want to search in directories within the specified directory. If you leave this option unselected, the CodeWarrior IDE searches only files at this level.

4. Use the **By type** drop-down menu to specify the type of file you want to search for.

5. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace, operation.

Searching through multiple files by projects

To search multiple files by projects:

1. Click the **In Projects** tab in the Find in Files dialog box (Figure 7-5).

Figure 7-5 In Projects tab, Find in Files dialog box

2. Use the **Project** drop-down menu to specify the project you want to search.

3. Use the **Target** drop-down menu to specify the target you want to search. Options are:
   - All Targets
   - Release
   - Debug.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select the required project-specific search options:

Project sources

Select this option to add all the source files from the current project. Deselect this option to remove all project source files from the file list.

System headers

Select this option to add system header files. System header files are defined in the CodeWarrior IDE Access Paths configuration panel. See *Configuring access paths* on page 10-12 for more information.

Note

- To search the system header files, you must have successfully compiled your source files.
- If this option does not add the header files you expect, use the `Make` command to update the internal list of header files. See *Making a project* on page 4-41 for more information.

Project headers

Select this option to add all the project header files from the current project. Deselect this option to remove all project header files from the file list.

Note

- To search the system header files, you must have successfully compiled your source files.
- If this option does not add the header files you expect, use the `Make` command to update the internal list of header files. See *Making a project* on page 4-41 for more information.

Search cached sub-targets

Select this option to search cached sub-targets of the projects shown in the **Project** drop-down menu. The search criteria that you define for the projects also apply to the cached sub-targets. If this option is disabled, cached sub-targets are disregarded during the search.

A cached sub-target has the following characteristics:

- the current target builds against it
- it resides in the current project
- it resides in a sub-project if you enabled sub-project caching for the current build target (see *Configuring build extras* on page 10-21 for more information).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

The file types you select are added to the search file list. See *Using file sets* on page 7-15 for information on saving sets of search files for future use.

Note

- To remove a specific file from the file list, select the file and press the backspace key.
- You can drag and drop groups or files from the Windows interface, or from the project window, onto the file list.

5. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace, operation.

Searching through multiple files by symbols

To search multiple files by symbols:

1. Click the **In Symbolics** tab in the Find in Files dialog box (Figure 7-6).

Figure 7-6 In Symbolics tab, Find in Files dialog box

2. Select the files containing symbolics information you want to search in the **Symbolics** drop-down menu. Selected files are added to the list at the bottom of the tab. To remove a file from the list, select it and press Delete. To open a file, double-click its name in the list.

3. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace, operation.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Searching through multiple files by file sets

To search multiple files by file sets:

1. Click the **In Files** tab in the Find in Files dialog box (Figure 7-7).

Figure 7-7 In Files tab, Find in Files dialog box

2. Select the file set you want to search in the **File Set** drop-down menu.
   - Select **Open Editor Files** if you want to view only files that are currently open.
   - Select the name of the appropriate file set if you want to search in a pre-saved file set.
   - Select **New File Set** if you want to create a new file set.

   See *Using file sets* on page 7-15 for information on file sets.

3. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace, operation.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7.3.2 Using file sets

You can use the **In Files** tab of the Find in Files dialog box to save sets of frequently searched files for later use, see:

- *Creating a file set*
- *Adding files to a file set*
- *Removing a file set* on page 7-16
- *Removing files from a file set* on page 7-16.

Creating a file set

To create a file set for use in future multifile searches:

1. Select **New File Set** from the **File Set** drop-down menu (see Figure 7-7 on page 7-14).
2. Click **Add Files…**.
3. Locate the required files and click **Add** to add them to the list.
4. Click **Save This Set...**. The CodeWarrior IDE displays the Save File Set dialog box (Figure 7-8).

Figure 7-8 Save File Set dialog box

5. Enter a name for the file set in the Save file set as text field.

Adding files to a file set

To add a file to an existing file set:

1. Select the required file set from the **File Set** drop-down menu.
2. Click the **Add Files...** button.
3. Locate the required files and click **Add** to add them to the list.

Note

You can also drag and drop files into the file set list to add them to the current list.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Removing a file set

To remove a file set from the list of saved search file sets:

1. Click the **Remove a Set** button. The CodeWarrior IDE displays the Remove File Sets dialog box (Figure 7-9).

Figure 7-9 Remove File Sets dialog box

2. Select the file set you want to remove and click **Remove**. The CodeWarrior IDE removes the file set.

3. Click **Done** to return to the Find in Files dialog box.

Removing files from a file set

To remove a file from a file set:

1. Select the required file set from the **File Set** drop-down menu.
2. Click on the required file to select it.
3. Press the Delete key.

To remove all the files from a file set:

1. Select the required file set from the **File Set** drop-down menu.
2. Click the **Clear List** button.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7.4 Using grep-style regular expressions

The CodeWarrior IDE provides regular expression searching that is similar to the UNIX `grep` command. A regular expression is a text string composed of characters, some of which have special meanings within the regular expression. The regular expression string describes one or more possible literal strings. In the CodeWarrior IDE Find dialog box, it is used to match literal strings in the search text if the **Regular expression** option is selected.

This section gives a brief introduction to regular expressions. For a comprehensive book on using regular expressions, see *Mastering Regular Expressions*, by Jeffrey E.F. Friedl.

This section describes:

- *Special operators*
- *Using regular expressions* on page 7-18.

7.4.1 Special operators

Table 7-1 shows the characters that have special meanings in a regular expression string. In some cases, their meaning depends on where they occur in the regular expression.

Table 7-1 Regular expression metacharacters

Metacharacter	Description
`.`	The *match-any-character operator* matches any single printing or non-printing character except `newline` and `null`.
`*`	The *match-zero-or-more* operator repeats the smallest preceding regular expression as many times as necessary (including zero) to match the pattern.
`+`	The *match-one-or-more* operator repeats the preceding regular expression at least once, and then as many times as necessary to match the pattern.
`?`	The *match-zero-or-one* operator repeats the preceding regular expression once or not at all.
`\n`	The *back-reference* operator is used in the replace string to refer to a specified group in the find string. Each group must be enclosed within parentheses. The digit `n` must range between 1 and 9. The number identifies a specific group, starting from the left side of the regular expression.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table 7-1 Regular expression metacharacters (continued)

Metacharacter	Description
|	The *alternation* operator matches one of a choice of regular expressions. If you place the alternation operator between any two regular expressions, the result matches the largest union of strings that it can match.
^	The *match-beginning-of-line* operator matches the string from the beginning of the string or after a newline character. When it appears within brackets, the ^ represents a *not* action.
$	The *match-end-of-line* operator matches the string either at the end of the string or before a newline character in the string.
[...]	*List* operators enable you to define a set of items to use as a match. The list items must be enclosed within square brackets. You cannot define an empty list.
(...)	*Group* operators define subexpressions that can be used elsewhere in the regular expression as a single unit.
-	The *range* operator defines the characters that fall between the start and ending characters within the list.

7.4.2 Using regular expressions

You can create powerful regular expressions to search for text and perform replace operations on found text. To use regular expressions in your search and replace strings:

1. Select **Find…** from the **Search** menu.
2. Ensure that the **Regular expression** option is selected (Figure 7-10).

Figure 7-10 Regular expression check box

3. Enter the search and replace strings. Your search and replace strings are treated as regular expressions.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 8
Working with the Browser

This chapter describes the CodeWarrior IDE browser which provides a user interface to access a database of all the symbols in your code quickly and easily. The CodeWarrior IDE browser works with both procedural and object-oriented code.

This chapter contains the following sections:

- *About working with the browser* on page 8-2
- *Enabling browser data production* on page 8-5
- *Using browser views* on page 8-8
- *Using the browser* on page 8-21
- *Creating classes and members with browser wizards* on page 8-30.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.1 About working with the browser

The CodeWarrior IDE browser enables you to view symbolic information for all C and C++ code in your project. The information is generated by the language parser, on the objects defined in your code, and the relationships between those objects. Browser information is generated in the background when a project is opened and is updated while you edit code. There is no need to compile the project.

Note

There is currently no support for generating browser data using the RealView Compiler. The language parser is used to generate browser data when the browser is activated and you build your project. This is the default setting for all ARM® stationery projects. See *Configuring build extras* on page 10-21.

Browser windows provide three *views* on the objects in the current build target, and enable you to navigate quickly to the source code for any object in the database. For example, you can find the function definition or declaration code for any member function of any class in your code.

Note

The browser does not distinguish between the declaration and the definition of a variable or constant, so the RealView® compiler produces a browse item for both. For example, the following code results in two items in the browse database:

```
// test.h
extern int var;
// test.c
#include test.h
int var;
```

You can use browser information both in browser windows, and from the CodeWarrior IDE editor. The browser is particularly useful for viewing object-oriented code, because it can map the relationships between classes, subclasses, and members. However it is also useful for navigating procedural code.

Note

It is not possible to generate browser data for assembler sources used in your project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.1.1 Understanding the browser strategy

The browser enables you to sort and examine information in a variety of ways. You can examine browser information using the following views:

Contents view

You can use the Contents view to view all C and C++ language constructs in your code, sorted by category into alphabetical lists. Categories include type definitions, constants, enumerations, macros, global variables, functions, templates, and classes. In addition, the Contents view lists assembler constructs such as register names, macros, and other symbols.

See *Viewing data by type with the Contents window* on page 8-17 for an example of a Contents view and details of the Contents view interface.

Class browser view

You can use the Class browser view to examine your code from a class-based perspective.

The Class browser view lists all the classes in your current build target, except classes that contain only data members. When you select a class in the list, the Class browser view displays its member functions and data members. When you select a list item, the source code where the item is defined is displayed in the Source pane.

See *Viewing data by class with the Class browser window* on page 8-8 for an example of a Class browser view and details of the user interface elements in the Class browser view.

Hierarchy view

The Hierarchy view is an inheritance-based view. It provides a graphical map of the class hierarchy for your current build target. You can use this view to follow class relationships.

The Hierarchy view illustrates how your classes are interconnected. You can expand and collapse a hierarchy from within this view.

See *Viewing class hierarchies and inheritance with the hierarchy view* on page 8-18 for an example of a Hierarchy view and details on the Hierarchy view interface.

The CodeWarrior IDE browser provides context-sensitive access to information. You can right-click on any symbol for which there is information in the database to display the related source code. See *Using the Browser context menu* on page 8-22 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

In addition, the CodeWarrior IDE browser enables you to decide the scope of the view. You can look at data in all your classes, or you can focus on one class.

Within the browser and hierarchy views, you can look at multiple class hierarchies or single class hierarchies. Table 8-1 summarizes the general viewing choices available when using the CodeWarrior IDE browser.

Table 8-1 Browser viewing options

Viewing style	Wide focus	Narrow focus
Comprehensive	Contents	Not applicable
Inheritance-based	Multi-class hierarchy	Single-class hierarchy

The browser-related menu commands in the **View** menu (**Browser Contents**, **Class Hierarchy**, and **Class Browser**) display wide-focus views. After you have selected a wide view, you can use a context-sensitive menu to focus on a particular class.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.2 Enabling browser data production

Browse data is not shared between targets. Therefore, you must enable the production of browser data for each target in your project. Browser data is enabled by default in all ARM stationery projects. To enable browser data production:

1. Ensure that your project window is the active window and click the **Target Settings** icon in the toolbar. The CodeWarrior IDE displays the Target Settings window for your project.

2. Click `Build Extras` in the Target Settings Panels list. The CodeWarrior IDE displays the Build Extras configuration panel (Figure 8-1).

Figure 8-1 Build Extras configuration panel

3. Select **Language Parser** from the drop-down menu in the Generate Browser Data from group box and click **Apply**. Browser data will now be generated in the background for this target.

4. Click on OK to close the Release Settings window.

See *Configuring browser options* on page 8-6 for more details on browser settings and options.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.2.1 Configuring browser options

Browser-related menu items and browser-specific options are available only when you enable the browser. See *Enabling browser data production* on page 8-5 for more information. This section describes how to configure additional browser options, including:

- *Configuring symbol colors*
- *Browsing across sub-projects*.

Note

To determine quickly if the browser is enabled, look in the **View** menu at the browser-related menu commands. If they are enabled, the browser is active.

Configuring symbol colors

You can use browser coloring to identify browser database symbols. If the browser is enabled, symbols that are in the browser database are displayed in editor and browser windows in the colors you select. See *Changing browser coloring* on page 9-36 for more information on setting browser colors.

Note

The default color setting is the same for all types of browser database symbols. You can select a different color for each symbol type if you want. However, if syntax coloring is also enabled for your code, you might find it easier to identify browser symbols if you use only one or two colors.

Browsing across sub-projects

To include browser information from sub-projects of the current build target you must enable sub-project caching.

Note

This option is selected by default.

To enable sub-project caching:

1. Ensure that your project window is the active window and click the **Target Settings** icon in the toolbar. The CodeWarrior IDE displays the Target Settings window for your project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click `Build Extras` in the Target Settings Panels list. The CodeWarrior IDE displays the Build Extras configuration panel (see Figure 8-1 on page 8-5).

3. Select the **Cache sub-projects** check box. This option:
   - improves multiproject updating and linking
   - enables the Class browser to include browser information from target sub-projects.

   However, this option also increases the amount of memory required by the CodeWarrior IDE.

   See *Configuring build extras* on page 10-21 for more information on the Build Extras configuration panel.

4. Click **Apply** to apply your changes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.3 Using browser views

This section describes how to use browser windows to display and work with data in the browser database. It describes:

- *Viewing data by class with the Class browser window*
- *Viewing data by type with the Contents window* on page 8-17
- *Viewing class hierarchies and inheritance with the hierarchy view* on page 8-18.

8.3.1 Viewing data by class with the Class browser window

The Class browser window provides a class-based view of the information in the browser database for the current build target. You can use the Class browser window to view C++ classes, member functions, and data members in the current build target.

Note

You must use the File Mappings configuration panel to map the ARM RealView Compiler to process all C++ source and header files containing classes. By default, header files are mapped to the ARM RealView Compiler.

Opening a Class browser window

To open the Class browser window:

1. Ensure that the browser is activated. See *Enabling browser data production* on page 8-5 for more information.
2. Select **Class Browser** from the **View** menu. Alternatively, you can:
   - right-click a class name to display the **Browser Contextual** menu and select **Open Browser for *classname***
   - double-click a class name in a Hierarchy window.

   The CodeWarrior IDE displays a Class browser window (see Figure 8-2 on page 8-9).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 8-2 Class browser window

The main components of the Class browser window are:

Browser toolbar

The browser toolbar provides access to a number of CodeWarrior IDE commands, including the **Go Forward** and **Go Back** buttons, and buttons to open Hierarchy view windows. See *Finding declarations, definitions, overrides, and multiple implementations* on page 8-25 for more information.

Browser Access Filters drop-down menu

Use this drop-down menu to filter the display of member functions and data members. See *Filtering members by access type* on page 8-15 for more information.

Pane zoom box

The pane zoom box enlarges and shrinks panes within the Class browser window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Resize bar A resize bar is located between each pair of panes. To resize two panes, drag the resize bar located between them.

Classes pane

The Classes pane lists classes in the browser database for the current build target. See *Viewing class and member information* on page 8-12 for more information.

List button Click the **List** button to toggle between an alphabetical list or a hierarchical list in the Classes pane. See *Viewing class and member information* on page 8-12 for more information.

Click this button to switch to a hierarchical list.

Click this button to switch to an alphabetical list.

Class display button

Click the **Class display** button at the bottom left of the Class browser window to toggle the display of the Classes pane.

Click this button to hide the classes pane.

Click this button to display the classes pane.

Class declaration button

Click the **Class declaration** button to display the class declaration for the current class in the Source pane. The name of the current class is displayed in the Status area of the Class browser window.

Member Functions pane

The Member Functions pane lists all member functions defined in the selected class. Constructors and destructors are at the top of the list. All other member functions are listed in alphabetical order.

To display inherited member functions select the **Show Inherited** check box in the toolbar.

The Inherited access icon in the Class browser window darkens to indicate that inherited member functions are currently displayed.

Note

Select a member function in the Member Functions pane in the Class browser window and press the Enter key to open an editor window and view the definition of the selected function.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Data Members pane

The Data Members pane lists all data members defined in the selected class. You can also display inherited data members by enabling the **Show Inherited** check box in the toolbar.

The Inherited access icon in the Class browser window darkens to indicate that inherited data members are currently displayed.

The entries in the Data Members pane are listed in alphabetical order. If inherited members are displayed, data members are listed by superclass, but alphabetically within each class.

Note

Select a data member in the Data Members pane in the Class browser window and press the Enter key to open an editor window and view the declaration of the selected data member.

Identifier icon

Member functions that are declared static, virtual, or pure virtual are identified with an icon. Table 8-2 describes the icons.

Table 8-2 Browser identifier icons

Icon	Meaning	The member is…
S	Static	A static member
V	Virtual	A virtual function that you can override, or an override of an inherited virtual function
P	Pure virtual	A member function that you must override in a subclass if you want to create instances of that subclass

Source pane

The Source pane displays the source code for the selected item.

Note

To enter function calls or variable names into the code in the source pane, Alt+click an item in the Member Functions pane or the Data Members pane. The item is entered into the Source pane text at the current insertion point.

The text in the Source pane is fully-editable. The path to the file that contains the code on display is shown at the top of the Source pane.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Open File icon

Click this icon to open the file that contains the code displayed in the Source pane in a new editor window.

VCS drop-down menu

The **VCS** drop-down menu is available if you have a version control system installed.

Note

For more information about the VCS plug-in modules which are available for use with the CodeWarrior IDE, see the Metrowerks web site.

Status area

The status area displays various status messages and other information. For example, when you select a class from the Classes pane, the status area displays the base classes for the selected class.

Viewing class and member information

The Class browser window enables you to locate and view class and member definitions in your source code. To view class and member information:

1. Open a Class browser window. See *Opening a Class browser window* on page 8-8 for more information.

   The class and member information is displayed in the panes of the Class browser window. The Classes pane see (Figure 8-3 on page 8-13) displays a list of classes in the current build target.

   Note

   The Classes pane does not display information about classes or structures that do not have any member functions, base classes, or subclasses. This means that structures and classes that have only fields and data members are not displayed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 8-3 Classes pane

2. Click the **List** button at the top right of the classes pane (see Figure 8-2 on page 8-9) to select either a hierarchical list or alphabetical list of classes in the classes pane:

   Alphabetical list

   This list type displays an alphabetical list of classes in the current build target.

   Hierarchical list

   This list type displays a hierarchy expansion triangle next to class names that have subclasses (Figure 8-3 shows an example of a hierarchical list):

   - Click an expansion triangle to toggle the display of subclasses.
   - Alt+click an expansion triangle to open all subclasses at all levels. This is called a *deep* disclosure.
   - Ctrl+click an expansion triangle to open a single level of subclass in a class and all of its siblings at the same level. This is a called a *wide* disclosure.
   - Ctrl+Alt+click an expansion triangle to perform a wide and deep disclosure.

   Note

   When you select a class in the Classes pane, the multi-class hierarchy window selection scrolls to the newly-selected class if it is not already displayed.

3. Navigate to the class, member function, or data member you want to view:

   a. Click within a pane to make it the active pane. You can also use the Tab key to navigate through the panes, except for the Source pane.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Caution

If the Source pane is active and you press the Tab key, a Tab character is entered into your source code.

b. Select an item within a pane in any of the following ways:

- Click an item in any list.
- Use the arrow keys to navigate through the items in the active pane.
- Type the name of the item. The item in the active pane that most closely matches the characters you type is selected.

When you select different items in the panes the Class browser window display changes:

- When you select a class name, the Member Functions pane and Data Members pane display the members of the selected class. Figure 8-4 shows an example for the class `Circle`.
- The source code pane displays the definition or declaration for the selected item. If the selected item is:
  - a class, the pane shows the class declaration
  - a function, the pane shows the function definition
  - a data member, the pane shows the data member declaration.

Figure 8-4 Member functions, data members, and declaration for class circle

Use the features of the browser to control the display of browser information, navigate to specific sections of code, or open other browser views. For more information on browser functions see:

- *Filtering members by access type* on page 8-15

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- *Opening another view from the Class browser view* on page 8-16
- *Using the Browser context menu* on page 8-22
- *Finding declarations, definitions, overrides, and multiple implementations* on page 8-25
- *Editing code in the browser* on page 8-29.

Filtering members by access type

You can use the **Access Filters** drop-down menu to filter the display of member functions and data members in the Class browser view. The drop-down menu commands filter the display according to public, private, and protected access types. To filter the display of members:

1. Open a Class browser window. See *Opening a Class browser window* on page 8-8 for more information.
2. Select the class you want to display. See *Viewing class and member information* on page 8-12 for more information.
3. Click on the **Access Filters** drop-down menu in the Class browser toolbar (Figure 8-5).

Figure 8-5 Access filters drop-down menu

The drop-down menu displays a list of access types. A bullet is displayed in the menu next to the access type currently selected.

4. Select the access type you want from the drop-down menu:

   View as implementor

   Select this option to show members with public, private, and protected access.

   View as subclass

   Select this option to show members with public and protected access.

   View as user

   Select this option to show only members with public access.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Show public

Select this option to show only members with public access

Show protected

Select this option to show only members with protected access.

Show private

Select this option to show only members with private access.

The access icons at the bottom right corner of the Class browser window are dark if the access type is selected, and grayed out if the access type is not selected (Figure 8-6).

Figure 8-6 Browser access filter icons

Opening another view from the Class browser view

There are a number of ways in which you can open a hierarchy or class view from the Class browser window, including:

- Click the **Show hierarchy** button to open a single-class or multi-class hierarchy window.
- Right-click any browser symbol in the window and use the **Browser** context menu. See *Using the Browser context menu* on page 8-22 for more information.

Saving a default Class browser window

You can save Class browser window configurations to be used as the default for new Class browser windows. You can save the:

- size and placement of Class browser window
- size and placement of the Classes, Member Functions, Data Members, and Source code panes within the Class browser window.

To save a default Class browser window:

1. Set up the Class browser window to your preferences. See *Opening a Class browser window* on page 8-8 for information on resizing controls in the Class browser window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select **Save Default Window** from the **Window** menu. The current Class browser window is saved and used as a default for all your CodeWarrior IDE projects.

8.3.2 Viewing data by type with the Contents window

The Contents window displays browser objects sorted by category into alphabetical lists.

Using the Contents window

To open a Contents window:

1. Ensure that the browser is activated. See *Enabling browser data production* on page 8-5 for more information.

2. Select **Browser Contents** from the **View** menu. The CodeWarrior IDE displays the Contents window (Figure 8-7). Alternatively, you can click the **Contents view** button in the Class browser toolbar, or use the **Browser** context menu.

Figure 8-7 Contents window

3. Select the category of data you want to view from the **Category** drop-down menu at the top of the window. The Symbols pane displays an alphabetical list of all symbols in the current build target for the selected category.

Note

Functions are listed alphabetically by function name, but the class name is displayed first. Therefore, it might appear that the functions are not listed alphabetically.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. From the Contents window you can:
   - right-click on any item in the contents list to display a **Browser** context menu for that item. See *Using the Browser context menu* on page 8-22 for more information.
   - double-click on any item in the contents list to open an editor window with the source code for the item.

8.3.3 Viewing class hierarchies and inheritance with the hierarchy view

You can use the browser hierarchy view to analyze inheritance in your source code. The hierarchy window displays a complete graphical map of the classes in the browser database for the current build target. Each class name is displayed in a box, and related classes are connected to each other by lines.

To open a hierarchy window and view the class hierarchy for the current build target:

1. Open the hierarchy window. There are three ways to do this:
   - Select **Class Hierarchy Window** from the **View** menu.
   - Click the **Show hierarchy** button in the Class browser window
   - Use the **Browser** context menu in the Contents window or the Class browser window. See *Using the Browser context menu* on page 8-22 for more information.

   The CodeWarrior IDE displays a hierarchy window for the current build target (see Figure 8-8 on page 8-19).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 8-8 Multi-class hierarchy window

2. In addition to the entry for each class, the main components of the hierarchy window are:

Line button

Click this button to toggle between diagonal lines and straight lines. This feature affects only the on-screen appearance of the hierarchy.

Hierarchy expansion triangle

Click this button to expose or conceal subclasses for a class.

- Click the expansion triangle to toggle the display of subclasses.
- Alt+click an expansion triangle to open all subclasses at all levels, that is, a deep disclosure.
- Ctrl+click an expansion triangle to open a single level of subclass in a class and all of its siblings at the same level, that is, a wide disclosure.
- Ctrl+Alt+click an expansion triangle to perform a wide and deep disclosure.

Note

Ctrl+Alt+click the expansion triangle for a base class that has no ancestors to expand or collapse an entire map.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Ancestor Class drop-down menu

If a class has multiple base classes, the hierarchy window displays a small triangle (the Ancestor Class triangle) to the left of the class name. Click on the Ancestor Class triangle to display the **Ancestor Class** drop-down menu. Figure 8-9 shows an example.

Select the ancestor class you want from the drop-down menu to jump to the hierarchy view for the ancestor class.

Figure 8-9 Ancestor Class drop-down menu

3. Navigate to the class you want to view:
   - Use the arrow keys to change the selected class:
     - use the Up and Down Arrow keys to move between siblings
     - use the Left and Right Arrow keys to move between ancestors and descendents.
   - Type the name of the class. The class selection changes to the closest match to the characters you type.
   - Use the Tab key to change the selected class alphabetically.

   Note

   The class selected in the hierarchy window changes when you select a class in the Classes pane of the Class browser window.

4. When you have selected a class you can:
   - double-click the class entry, or select the entry and press the Enter key to open a Class browser window for that class. *Viewing data by class with the Class browser window* on page 8-8 for more information.
   - right-click on the class to open a **Browser** context menu. See *Using the Browser context menu* on page 8-22 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.4 Using the browser

This section gives some techniques you can use to perform common tasks with the browser. It describes:

- *Using Go Back and Go Forward*
- *Using the Browser context menu* on page 8-22
- *Finding declarations, definitions, overrides, and multiple implementations* on page 8-25
- *Using symbol name completion* on page 8-28
- *Editing code in the browser* on page 8-29.

8.4.1 Using Go Back and Go Forward

Use the **Go Back** and **Go Forward** commands to retrace your navigational steps through source code and browser views. Either:

- Select **Go Back** or **Go Forward** from the **Search** menu to move back or forward one view.
- Click the **Go Back** or **Go Forward** buttons in the browser toolbar to move back or forward one view.
- Click and hold the **Go Back** and **Go Forward** buttons to display a drop-down menu containing a list of previous views (Figure 8-10).

Figure 8-10 Go Back and Go Forward toolbar buttons

Note

Go Back and **Go Forward** do not undo any actions you performed.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.4.2 Using the Browser context menu

The **Browser** context menu is a context-sensitive menu that provides quick access to browser information. The **Browser** context menu is available for any symbol for which the browser database has data. You can use it to access the source code related to any symbol. To display the **Browser** context menu:

1. Ensure that the browser is activated. See *Enabling browser data production* on page 8-5 for more information.
2. Open any of the Class, Contents, or Hierarchy browser windows. See *Using browser views* on page 8-8 for more information. You can also open a source code file in the CodeWarrior IDE editor.
3. Right-click on any symbol name in the window. The **Browser** context menu is displayed. The menu commands available in the menu depend on the symbol type (such as class, function name, and enumeration), and the context in which the menu was called. Figure 8-11 shows an example of a **Browser** context menu for a member function.

Figure 8-11 Browser context menu for a function

For member functions, you can:

- View the function declaration. See *Viewing a class or member declaration* on page 8-25.
- View the function definition. See *Viewing a function definition* on page 8-25.
- Use the **Find all implementations of** command to find all implementations of a function that has multiple definitions. See *Finding overrides and multiple implementations of a function* on page 8-26.

Using the Browser context menu from an editor window

In the CodeWarrior IDE editor window, every symbol in your code, such as function names, class names, data member names, constants, enumerations, templates, macros, and type definitions, becomes a hypertext link to other locations in your source code. For example, you can right-click on a class name to:

- open the class declaration
- open a Class browser window for that class
- open a Hierarchy window for that class.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

For function names, you can use the **Browser** context menu to insert function templates into your code. See *Using the Insert template commands* on page 8-24 for more information.

Note

- The **Browser** context menu displays a **Set Breakpoint** menu item in source code windows. This command is not implemented in the ARM version of the CodeWarrior IDE.
- The contextual menu features of the browser work with the CodeWarrior IDE editor, in addition to all browser windows. For this reason, you should consider enabling the browser, even if you do not use the browser windows.

You can use symbol name completion to enter a browser symbol into your text file:

1. Select the text and right-click to display the **Browser** context menu. The menu contains a list of browser symbols that match part or all of the selected text. Figure 8-12 shows an example for the character string `bm`, where `bmw` and `bmw_h` are both symbols in the browser database.

Figure 8-12 Using symbol name completion

2. Select an item from the list to enter it into your text file. See *Using symbol name completion* on page 8-28 for other ways to type browser items.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Using the Insert template commands

You can use the context-sensitive menu in an editor window to insert function templates into your code. To insert a function template for a specific function:

1. Ensure that the **Insert Template Commands** option is selected in the Editor Settings configuration panel. See *Editor settings* on page 9-24 for more information. This option is off by default.

2. Type the name of the function you want to insert, and right-click. If the function has one or more definitions, the **Browser** context menu displays **Insert** commands for each definition. Figure 8-13 shows an example.

Figure 8-13 Inserting a function template

3. Select the function template you want to insert. The CodeWarrior IDE inserts template code for the function.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.4.3 Finding declarations, definitions, overrides, and multiple implementations

This section describes how to use the browser to navigate through your source code. It describes:

- *Viewing a class or member declaration*
- *Viewing a function definition*
- *Finding overrides and multiple implementations of a function* on page 8-26.

Viewing a class or member declaration

Use any of the following methods to display a class or member declaration:

- Select a class name or data member name in a Class browser window. The declaration is displayed in the Source pane. Double-click the name to open the file that contains the declaration. (If you select or double-click a function name, the function definition is displayed.)
- Click the **Class Declaration** button in the Class browser window to display a class declaration in the Source pane.
- Right-click on a class or member name in any editor or browser window and select **Go to declaration of *name*** from the **Browser** context menu to jump to the declaration.

Viewing a function definition

Use any of the following methods to display a function definition:

- Select the function in the Member Functions pane of the Class browser window. The definition is displayed in the Source pane. To open the file that contains the definition, double-click the function name in the Member Functions pane.
- Right-click on the function name in any editor or browser window and select **Go to definition of *name*** from the **Browser** context menu to jump to the function definition.
- Alt+double-click or Ctrl+double-click a function name in any source view. The Symbol window is displayed for functions with multiple definitions to show all implementations of that function. Figure 8-14 on page 8-26 shows an example for the symbol `g`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 8-14 Symbol window for a multiply defined function

- Right-click on a class name in any browser window. The **Browser** context menu displays a list of member functions, if any are defined for the class. Use the menu to jump to the function definition.

Finding overrides and multiple implementations of a function

The Symbol window lists all implementations of any symbol that has multiple definitions. Typically, these symbols are multiple versions of overridden functions. However, the Symbol window works for any symbol that has multiple definitions in the browser database.

To list implementations of a symbol:

1. Find an instance of the symbol name in any browser or editor window. For example, to find overrides of virtual functions, open a Class browser window and look for functions that are marked with a virtual identifier icon . These are either:
   - overrides of inherited virtual functions
   - virtual functions declared in the class that are not inherited from an ancestor.
2. Right-click on the symbol name. A **Browser** context menu is displayed.
3. Select **Find all implementations of *symbol_name*** from the **Browser** context menu. The CodeWarrior IDE displays the Symbol window with a list of all definitions for the symbol (Figure 8-15 on page 8-27).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

In a source pane or editor window, Alt+double-click or Ctrl+double-click a function or other symbol name to find all implementations, and open the Symbol window without using the contextual menu.

Figure 8-15 Symbol window

Most of the items in the Symbol window work in the same way as the corresponding items in the Class browser window. See *Viewing data by class with the Class browser window* on page 8-8 for more information. The Symbol window has two items which are not found in the Class browser window:

Symbols pane

This pane lists all versions of a symbol in the database.

Select an item in the Symbols pane to display its definition in the Source pane.

Orientation button

Click this button to toggle the orientation of the Symbols pane and the Source pane.

4. Select an implementation in the Symbol window list to display its definition in the source pane.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.4.4 Using symbol name completion

Use the following keyboard commands to find and select browser items that match the text you have selected or just typed into a source code file.

Note

The following commands are available only from the keyboard. They are not available in the CodeWarrior IDE menus.

Find symbols with prefix

Type Ctrl+\ to enter the name of a browser item that has the same initial characters as the text you have selected or just typed.

Find symbols with substring

Type Ctrl+Shift+\ to enter the name of a browser item that has a substring with the same characters as the text you have selected or just typed.

Get next symbol

Type Ctrl+. (Ctrl+period) after using one of the **Find symbols** commands to search for the next symbol in the database that matches your search string.

Get previous symbol

Type Ctrl+, after using one of the **Find symbols** commands to search for the previous symbol in the database that matches your search string.

When you find the browser item you want to enter, press the Right Arrow key to place the insertion point next to the item and continue typing.

Note

Another way to find and enter a browser item is to right-click on the first few characters of the text and wait for the **Browser** context menu to display. The menu displays a list of matching items. Select an item to enter it into your text. See *Using the Browser context menu* on page 8-22 for more details.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.4.5 Editing code in the browser

Code displayed in a Source code pane is fully editable. You can use standard CodeWarrior IDE editor commands to edit your code (see Chapter 6 *Editing Source Code* for more information).

Opening a source file

Use any of the following methods to open a source file:

- In the Class browser window, click the **Open File** button when the file is displayed in the Source pane (see Figure 8-2 on page 8-9).
- Right-click on a symbol used in the source file and use the **Browser** context menu to open the file.
- Type Ctrl+` to move between a source file and its corresponding header file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.5 Creating classes and members with browser wizards

When you open a browser window, the CodeWarrior IDE adds a **Browser** menu to the main menu bar. You can use the commands in the **Browser** menu to display browser wizards that help you create new classes, member functions, and data members.

Note

The wizards assume that you have a basic understanding of C++.

The commands in the **Browser** menu that are implemented by the ARM version of the CodeWarrior IDE are:

New Class…
Displays the New Class wizard to help you create a new class. You can specify the name, location, file type, and modifiers for the new class. See *Using the New Class wizard* on page 8-31.

New Member Function…
Displays the New Member Function wizard to help you create a new member function for a selected class. You can specify the name, return type, parameters, modifiers, and other optional information for the new member function. See *Using the New Member Function wizard* on page 8-35.

New Data Member…
Displays the New Data Member wizard to help you create a new data member for a selected class. You can specify the name, type, initializer, modifiers, and other optional information for the new data member. See *Using the New Data Member wizard* on page 8-38.

Note

The **New Property…**, **New Method…**, **New Event Set…**, and **New Event…** menu items are not implemented by the ARM version of the CodeWarrior IDE.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.5.1 Using the New Class wizard

You can use the New Class wizard to create a new class declaration, or a class declaration based on an existing class:

1. Ensure that one of the browser windows is the currently active window. See *Using browser views* on page 8-8 for information on opening browser windows.

2. Select **New Class…** from the **Browser** menu. The CodeWarrior IDE displays the New C++ Class: Name and Location dialog box (Figure 8-16).

Figure 8-16 New Class wizard: Name and Location

3. Enter the name and location for the new class:

Class Name

Enter a name for the new class. The wizard names the declaration and definition files depending on the values you specify for the options listed below.

Declaration File

Use this drop-down menu to specify the type of declaration file. Depending on the option you choose, different fields become enabled below the **Declaration File** drop-down menu. You can select either:

New File Select this option to create a new declaration file. Enter the path name for the new file, or click **Set…** to use the standard file dialog box to set a directory for the new file. By default the file is saved with the name `classname.h`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Relative to class

Select this option to add the class to an existing declaration file. Enter the name of an existing class where you want to declare the new class, or click **Set…** to select a class from a list of current classes in the browser database. Use the drop-down menu to place the new class **Before** or **After** the selected class declaration.

Namespace

Namespaces are not supported by the RealView Compiler. Leave this field empty.

Use separate file for member definitions

Select this check box if you want to use a separate file to define the members of the new class. Type the path to the separate file in the field below the check box, or click **Existing...** to select the file with the standard file dialog box. To create a new separate file, click **New...** and save the new file to a location on your hard disk.

4. Click **Next >** to move to the next page of the New Class wizard. The Base Classes and Methods dialog box is displayed (Figure 8-17).

Figure 8-17 New C++ Class: Base Classes and Methods dialog box

5. Specify base classes, member functions, and other information for the new class:

Base Classes

Enter a comma-separated list of base classes for the new class.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Generate Constructor and Destructor

Select this check box to generate a constructor and destructor for the new class. The following options are available:

Access Select an access type for the constructor and destructor from the drop-down menu.

Constructor parameters

Enter a list of parameters for the constructor. Example parameters are listed above the field.

Virtual destructor

Select this check box to create a virtual destructor for the new class.

Namespaces

Enter any namespaces required for the base classes or constructor parameters you are adding.

6. Click **Next >** to move to the next page of the New Class wizard. The New C++ Class: Include Files dialog box is displayed (Figure 8-18).

Figure 8-18 New C++ Class: Include Files dialog box

7. Enter a list of any additional `#include` files for the new class. Separate each file in the list with a comma. The `#include` files that are added automatically are listed in the field above.

8. Click **Next >** to move to the next page of the New Class wizard. The New C++ Class: Targets dialog box is displayed (Figure 8-19 on page 8-34).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 8-19 New C++ Class: Targets dialog box

9. Select the check box next to one or more build targets to assign the new class to the build targets you want. You must select at least one build target.

10. Click **Finish**. The CodeWarrior IDE displays a summary of the class information you have specified (Figure 8-20).

Figure 8-20 New Class wizard, summary information

11. Click **Cancel** to return to the New Class wizard to change your settings.

12. Click **Generate** to create the new class and close the New Class wizard.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.5.2 Using the New Member Function wizard

You can use the New Member Function wizard to create a new member function for an existing class:

1. Ensure that one of the browser windows is the currently active window. See *Using browser views* on page 8-8 for information on opening browser windows.
2. Select the class to which you want to add the member function. For example, in the Class browser window, click on the class name in the Classes list at the left of the window.
3. Select **New Member Function…** from the **Browser** menu. The CodeWarrior IDE displays the Member Function Declaration dialog box (see Figure 8-21).

Figure 8-21 New C++ Member Function: Member Function Declaration dialog box

4. Enter information for the new member function declaration:

   Name Enter the name for the member function.

   Return type
   Enter the function return type.

   Parameters
   This field is optional. Enter a comma-separated list of parameters for the member function, if required.

   Namespaces required for parameters (optional)
   Enter any namespace required for the parameters you are adding.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Modifiers

Modify the function declaration, as required:

- Use the **Access** drop-down menu to specify whether the new member function is public, protected, or private.
- Use the **Specifier** drop-down menu if you want to declare the new member function as a virtual, pure virtual, or static function.
- Select the **Inline** or **Const** options to declare the function `inline` or `const`. If you select the **Inline** check box, the CodeWarrior IDE places the framework function definition within the class.

5. Click **Next >** to move to the next page of the New Member Function wizard. The New C++ Member Function: File Locations dialog box is displayed (Figure 8-22).

Figure 8-22 New C++ Member Function: File Locations dialog box

6. Specify file locations for the new member function:

Declaration

This field displays the location of the file to which the member function declaration will be added.

Definition

Enter the path to the file used for the member function definition or click **Existing…** to select the file using a standard file dialog box. To create a new file to use for the member function definition, click **New…** and save the new file to a location on your hard disk.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Include files

This field displays a list of `#include` files that will be automatically added to the member function. These files are automatically added based on the return type and parameters you specified from the previous section.

Additional header include files

Enter a list of any additional `#include` files you require for the new member function.

7. Click **Finish**. The CodeWarrior IDE displays a summary of the information you have entered for the new member function declaration (Figure 8-23).

Figure 8-23 New Member Function wizard, summary information

8. Click **Cancel** to return to the New Member Function wizard to change your settings.

9. Click **Generate** to generate source for the new member function. The CodeWarrior IDE adds a member function declaration to the selected class, and creates a framework function definition below the class, or inline if the **Inline** check box is selected.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

8.5.3 Using the New Data Member wizard

You can use the New Data Member wizard to create a new data member declaration in an existing class:

1. Ensure that one of the browser windows is the currently active window. See *Using browser views* on page 8-8 for information on opening browser windows.
2. Select the class to which you want to add the data member. For example, in the Class browser window, click on the class name in the Classes list at the left of the window.
3. Select **New Data Member…** from the **Browser** menu. The CodeWarrior IDE displays the New C++ Data Member: Data Member Declaration dialog box (see Figure 8-24).

Figure 8-24 New C++ Data Member: Data Member Declaration dialog box

4. Declare the new data member:

   Name Enter a name for the data member.

   Type Enter the data member type.

   Namespaces required for type (optional)

   Enter any namespace required for the data member types you are adding.

   Initializer

   Type an initial value for the data member. This field is optional.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Modifiers

Use the **Access** and **Specifier** drop-down menus to select the access level and member specifier for the new data member. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers include **None**, **Static**, and **Mutable**. Enable the **Const** or **Volatile** check boxes as required to further describe the modifiers of the data member.

5. Click **Next >** to move to the next page of the New Data Member wizard. The New C++ Data Member: File Locations dialog box is displayed (see Figure 8-25).

Figure 8-25 New Data Member wizard: File Locations

6. Specify file locations for the new data member:

Declaration

This field displays the location of the file to which the data member declaration will be added.

Definition

This field does not apply to data members.

Include file automatically added for member type

This field displays any `#include` files automatically added for the data-member type.

Additional header include files

Enter a list of any additional `#include` files you require for the new data member.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

7. Click **Finish**. The CodeWarrior IDE displays a summary of the information you have entered for the new data member (see Figure 8-26).

Figure 8-26 New Data Member wizard, summary information

8. Click **Cancel** to return to the New Data Member wizard to change your settings.

9. Click **Generate** to generate source for the new data member. The CodeWarrior IDE adds a data member declaration to the selected class.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 9
Configuring IDE Options

This chapter describes how to set options in the CodeWarrior IDE Preferences window. In addition, this chapter describes how to configure the CodeWarrior IDE toolbars and key bindings for commands. It contains the following sections:

- *About configuring the CodeWarrior IDE* on page 9-2
- *Overview of the IDE Preferences window* on page 9-3
- *Choosing general preferences* on page 9-6
- *Choosing editor preferences* on page 9-21
- *Setting commands and key bindings* on page 9-37
- *Customizing toolbars* on page 9-48.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.1 About configuring the CodeWarrior IDE

You can use the IDE Preferences window to customize many features of the CodeWarrior IDE. The settings you specify in this window are global settings. They affect the way the CodeWarrior IDE works in all projects. You can export your settings to XML and import an XML settings file. In addition, you can customize toolbars and commands to suit your own working style.

This chapter describes:

Setting general preferences

General preferences enable you to customize a number of features of the CodeWarrior IDE, including build settings and global source trees. You can also export settings to XML and import an XML settings file. See *Choosing general preferences* on page 9-6 for more information.

Setting editor preferences

You can use the Editor preference panels to set many options that affect how you edit text, including the number of items in the **Open Recent** submenu, syntax coloring, and font and tabs settings. In addition you can specify a third-party editor to be used in place of the CodeWarrior IDE editor. See *Choosing editor preferences* on page 9-21 for more information.

Customizing commands and key bindings

You can customize the menu commands that are displayed in the CodeWarrior IDE, and the keyboard shortcuts that are assigned to menu commands. See *Setting commands and key bindings* on page 9-37 for more information.

Customizing toolbars

You can customize the items that are displayed as icons in the CodeWarrior IDE toolbars. You can create toolbar icons for most menu commands, and add interface elements to the toolbar. See *Customizing toolbars* on page 9-48 for more information.

Note

The Debugger and RAD Tools configuration panels, which appear in the IDE Preferences window, are not used by the ARM® version of the CodeWarrior IDE.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.2 Overview of the IDE Preferences window

This section gives an overview of how to use the IDE Preferences window to configure global preferences for the CodeWarrior IDE. Detailed instructions on how to set specific preferences are described in the sections that follow this overview.

9.2.1 Using the IDE Preferences window

This section gives basic information on using the IDE Preferences window to configure preferences for all your CodeWarrior IDE projects.

Opening the IDE Preferences panels

To open the IDE Preferences window and select preferences:

1. Select **Preferences…** from the **Edit** menu. The CodeWarrior IDE displays the IDE Preferences window with a hierarchical list of available panels on the left side of the window. Figure 9-1 shows an example.

Figure 9-1 IDE Preferences window

2. Select the configuration panel you want to configure from the list. You can use the arrow keys or click the name of the panel.

   Each configuration panel contains related options that you can set. The options you select apply to all CodeWarrior IDE projects.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Select the options you require. See the following sections in this chapter for detailed descriptions of the options in each configuration panel.

4. Apply or cancel your changes, as required. See *Saving or discarding changes* for more information on applying the changes you have made.

Saving or discarding changes

You can use the dialog box buttons in the IDE Preferences window to apply or discard your changes (see Figure 9-1 on page 9-3):

OK
Click this button to save any changes that you have made and close the IDE Preferences window.

Cancel
Click this button to close the window without saving any changes you have made.

Apply
Click this button to commit any changes you have made in any of the panels. If you have changed an option that requires you to recompile the project, the CodeWarrior IDE displays a confirmation dialog box. Click **OK** or **Cancel** depending on whether you want to keep your changes or not.

Factory Settings
Click this button to reset the current panel to the settings that the CodeWarrior IDE uses as defaults. Settings in other panels are not affected. Only the settings for the current panel are reset.

Revert
Click this button to reset the state of the current panel to its last-saved settings. This is useful if you start making changes to a configuration panel and then decide not to use them.

Export Panel...
Click this button if you want to export the current settings for a panel to XML format. Enter the name of the file to which you want to export the settings and click the **Save** button. You can then import these settings at any time.

Import Panel...
Click this button if you want to import an XML settings file created by using the **Export Panel...** option. Select the file you want to import and click **Open** to import the settings.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

The **Import Panel...** and **Export Panel...** options are useful if you want to copy *all* the settings for a configuration panel to the same panel in another target. If you only want to copy some panel settings, for example, the settings for one tab in a panel, you must edit the panel manually.

If you make changes in the IDE Preferences window and attempt to close it, the CodeWarrior IDE displays a preferences confirmation dialog box (Figure 9-2).

Figure 9-2 Confirmation dialog box

Click:

- **Save** to save your changes and close the dialog box.
- **Don't Save** to discard your changes and close the dialog box.
- **Cancel** to continue using the IDE Preferences window without saving changes.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.3 Choosing general preferences

This section describes how to set preferences for the CodeWarrior IDE as a whole, including editor preferences. The preferences you set apply to all your CodeWarrior IDE projects.

This section describes:

- *Configuring build settings*
- *Configuring concurrent compiles* on page 9-8
- *Configuring IDE extras* on page 9-9
- *Configuring plugin settings* on page 9-12
- *Configuring shielded folders* on page 9-14
- *Configuring global source trees* on page 9-17.

9.3.1 Configuring build settings

The Build Settings configuration panel enables you to customize a number of project build settings. To open the Build Settings configuration panel:

1. Select **Preferences…** from the **Edit** menu and click `Build Settings` in the IDE Preference Panels list. The CodeWarrior IDE displays the Build Settings configuration panel (Figure 9-3).

Figure 9-3 Build Settings configuration panel

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Change the following options, as required:

Build before running

Use this drop-down menu to configure how the CodeWarrior IDE responds if you try to run a project and the source or target settings for the project have been changed since the last build. You can select from:

Always Always build changed projects before running them.

Ask The CodeWarrior IDE will ask you how to proceed if you have changed the project since the last build.

Never Never build changed projects before running them.

Save open files before build

Select this option if you want to save all open files automatically before a **Preprocess**, **Compile**, **Disassemble**, **Bring Up To Date**, **Make**, **Run** or **Debug** command is executed.

Show message after building up-to-date project

Select this option to configure the CodeWarrior IDE to display a message when you try to build an up-to-date project. An up-to-date project is not rebuilt.

If this option is not selected, the CodeWarrior IDE does nothing when you try to build an up-to-date project.

Compiler thread stack (K)

This option is not implemented by the ARM version of the CodeWarrior IDE.

Use Local Project Data Storage

After loading a project file, the CodeWarrior IDE creates or updates an associated project data directory where it stores intermediate project data. After you build or close the project, the CodeWarrior IDE uses the information in the project data directory to update the project file.

By default, the CodeWarrior IDE places the project data directory within the same directory as the project file. However, the CodeWarrior IDE cannot create or update a project data directory in a location that grants read-only privileges. To specify a different location for the project data directory, select the **Use Local Project Data Storage** check box, then click **Choose...** to select the path to the location you want to use. For more information on the types of paths available to you, see *Configuring global source trees* on page 9-17.

3. Click **Apply** to apply your changes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.3.2 Configuring concurrent compiles

Use the Concurrent Compiles configuration panel to enable concurrent compilations for your project. When this feature is enabled, the CodeWarrior IDE makes better use of available processor capacity by spawning multiple compile processes to compile code more efficiently. When concurrent compilations are enabled, compilation is improved by:

- optimizing resource use
- using overlapped input/output.

Note

You must have enough ARM licenses available for the number of concurrent compilations that you want to use.

To open the Concurrent Compiles configuration panel:

1. Select **Preferences…** from the **Edit** menu and click `Concurrent Compiles` in the IDE Preference Panels list. The CodeWarrior IDE displays the Concurrent Compiles configuration panel (Figure 9-4).

Figure 9-4 Concurrent Compiles configuration panel

2. Click **Use Concurrent Compiles** to enable concurrent compiles.

3. Click on **Recommended** or **User specified** to specify the number of compilation processes which can be run simultaneously.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Click **Apply** to apply your changes.

9.3.3 Configuring IDE extras

The IDE Extras configuration panel has options to remember previously-opened projects and text files, and enables you to configure the CodeWarrior IDE to use third-party editors. To open the IDE Extras configuration panel:

1. Select **Preferences…** from the **Edit** menu and click `IDE Extras` in the IDE Preference Panels list. The CodeWarrior IDE displays the IDE Extras configuration panel (Figure 9-5).

Figure 9-5 IDE Extras configuration panel

There are three groups of options. For details of how to change the options see:

- *Configuring menus*
- *Using a third-party text editor* on page 9-10
- *Configuring other settings* on page 9-11.

Configuring menus

Use the **Menu bar layout** drop-down menu to specify the default menu layout for the CodeWarrior IDE. The name of the menu bar layout suggests the host. For example, use the Windows menu bar layout to organize the CodeWarrior IDE menus according to Windows user interface standards.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

The CodeWarrior IDE for RVDS does not use the Macintosh layout, and only the Windows layout is documented in this book.

You can configure how many projects and documents are displayed in the **Open Recent** submenu of the **File** menu. To set the number of project and documents displayed:

1. Open the IDE Extras configuration panel (see *Configuring IDE extras* on page 9-9).
2. Enter values for the following text fields:

   Recent projects
   Enter the maximum number of projects you want displayed in the **Open Recent** submenu.

   Recent documents
   Enter the maximum number of files you want displayed in the **Open Recent** submenu.

   Recent symbolics
   Enter the maximum number of symbolics you want displayed in the **Open Recent** submenu.

   Note

   CodeWarrior IDE for RVDS does not use symbolic files.

   Recent workspace
   Enter the maximum number of workspaces you want displayed in the **Open Recent** submenu.
3. Click **Apply** to apply your changes.

Appendix D *CodeWarrior IDE Reference* shows the organization of the menu commands under each menu bar layout.

Using a third-party text editor

You can configure the CodeWarrior IDE to use a third-party text editor in place of its built-in text editor. To use a third-party editor:

1. Open the IDE Extras configuration panel (see *Configuring IDE extras* on page 9-9).
2. Select the **Use Third Party Editor** check box. When this check box is selected, the CodeWarrior IDE uses the third-party text editor you specify to open text files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Enter the command line to invoke the text editor:
   a. Type the name of the editor you want to use in the Launch Editor text field.
   b. Type the name of the editor and an initial line of text to jump to on launch in the Launch Editor w/Line # text field. The CodeWarrior IDE invokes this command line when you double-click on an error message to display the line in the text file that caused the error message.

   You can use two variables to specify the file you want to open, and the line you want to jump to:

   `%file` The CodeWarrior IDE expands this into the full path name of the file.

   `%line` The CodeWarrior IDE expands this into the initial line number for the file.

   For example, to use the Emacs text editor to edit text files, type:

   ```
   runemacs %file
   ```

   into the Launch editor text field, and type:

   ```
   runemacs +%line %file
   ```

   into the Launch Editor w/Line # text field.

   See your text editor documentation for more information on specifying line numbers.

   Note

   The CodeWarrior IDE does not recognize that files have been modified in a third-party editor if the **Use modification date caching** option is selected. See *Configuring build extras* on page 10-21 for more information.

4. Click **Apply** to apply your changes.

Configuring other settings

The Other Settings group contains options that enable you to configure various user interface and display features of the CodeWarrior IDE:

- Use Multiple Document Interface (MDI)
- Use default workspace
- Show Code and Data Sizes.

To use MDI:

1. Open the IDE Extras configuration panel (see *Configuring IDE extras* on page 9-9).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Select the **Use Multiple Document Interface** check box to use the Windows *Multiple Document Interface* (MDI).

   Deselect the check box to use the *Floating Document Interface* (FDI).

3. Click **Apply** to apply your changes. You must quit and restart the CodeWarrior IDE to apply your changes.

For more information about MDI and FDI, see *Working with project windows* on page 3-18.

To use the default workspace:

1. Open the IDE Extras configuration panel (see *Configuring IDE extras* on page 9-9).

2. Select the **Use default workspace** check box. When this check box is selected the state of all open CodeWarrior IDE windows will be retained when you exit the CodeWarrior IDE. Deselect this option to open the CodeWarrior IDE with no windows visible.

For more information about workspaces, see *Working with project workspaces* on page 3-22.

To show code and data sizes:

1. Open the IDE Extras configuration panel (see *Configuring IDE extras* on page 9-9).

2. Select the **Show Code and Data Sizes** check box. When this check box is selected code and data sizes for project files are displayed in the project window.

For more information about displaying information in the project window, see *Project window, Files view* on page 3-5.

9.3.4 Configuring plugin settings

Use the plugin Settings configuration panel to specify how much plug-in diagnostic information the CodeWarrior IDE provides. Plug-in diagnostic information is useful if you are using the CodeWarrior IDE to develop plug-ins for the CodeWarrior IDE. Use this panel if you have problems getting your plug-in to function properly, or if you want more information about the properties of installed plug-ins.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

You cannot develop CodeWarrior IDE plug-ins with RVDS tools. However, if you develop a CodeWarrior IDE plug-in using the standard Metrowerks CodeWarrior development environment and the plug-in SDK, you can use this option to diagnose problems when you run the plug-in from the CodeWarrior environment.

To set plug-in diagnostics:

1. Select **Preferences…** from the **Edit** menu and click **Plug-in Settings** in the **IDE Preference Panels** list. The CodeWarrior IDE displays the **Plugin Settings** configuration panel (Figure 9-6).

Figure 9-6 Plugin Settings configuration panel

2. Select the level of plug-in diagnostics you want:

   None Select this setting if you do not want to generate plug-in diagnostics. This is the default setting. No plug-in diagnosis takes place, and no output is produced.

   Errors Only
   Select this setting to display errors that occur when the CodeWarrior IDE loads plug-ins. The errors are displayed in a new text document after the CodeWarrior IDE starts up. You can save or print the text file after it is generated so you can have a convenient error reference when troubleshooting your plug-ins.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

All Info Select this setting to display detailed information for each plug-in. Problems with loading plug-ins, optional plug-in information, and plug-in properties are reported. This information is displayed in a new text document after the CodeWarrior IDE starts up.

The text document includes a complete list of installed plug-ins and their associated preference panels, compiler, and linkers, and provides suggestions for correcting general plug-in errors. You can save or print the text file after it is generated so you can have a convenient error reference when troubleshooting your plug-ins.

3. Click **Disable third party COM plugins** if you want to disable all third-party *Common Object Model* (COM) plug-ins for the CodeWarrior IDE. This can be helpful when you are troubleshooting.

   Note

   If you experience problems when using the CodeWarrior IDE, you should select this option. If the problem no longer occurs after disabling third-party plug-ins, this indicates that the source of the problem might be conflicts between third-party plug-ins and the plug-ins supplied with the CodeWarrior IDE.

4. Click **Apply** to apply your changes. The CodeWarrior IDE warns that you must quit and restart the CodeWarrior IDE for the changes to take effect. Plug-in diagnostics are generated when you restart the CodeWarrior IDE.

9.3.5 Configuring shielded folders

The **Shielded Folders** configuration panel enables you to configure the CodeWarrior IDE to ignore specified directories during certain operations. This option is useful, for example, if you want to avoid collisions between example files or version control data. You specify regular expressions that match the names of the directories that you want to ignore. The CodeWarrior IDE treats matching directories as shielded folders and ignores their contents. For more information about regular expressions, see *Using grep-style regular expressions* on page 7-17.

The **Shielded Folders List** shows the current list of regular expressions. The CodeWarrior IDE uses this list to determine whether to skip directories during project operations or find-and-compare operations. The following regular expressions appear in the **Shielded Folders List** by default:

- The `\(.*\)` regular expression matches directories with names that begin and end with parentheses, such as `(Project Stationery)`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- The `CVS` regular expression matches directories named CVS. With this regular expression, the CodeWarrior IDE skips any data files used by the CVS version control system
- The `.*[_]Data` regular expression matches the names of directories used by the CodeWarrior IDE to store target data information, such as `MyProject_Data`.

The **Shielded Folders List** includes two columns. These columns represent different types of operations against which you can shield folders:

- A bullet in the Project operations column indicates that the CodeWarrior IDE ignores matching directories for project operations. Such operations include dragging a directory into the project window, building a project, and searching access paths for source or include files after choosing the Open command.

- A bullet in the Find-and-compare operations column indicates that the CodeWarrior IDE ignores matching directories for find-and-compare operations. These operations include dragging a directory into fields in the Find window, or comparing directory contents.

Note

You can configure a project target to include items within a shielded folder. Use the Access Paths target configuration panel to specify the path to the shielded folder containing files you want to include. See *Configuring access paths* on page 10-12 for more information on configuring access paths.

To add a regular expression to the **Shielded Folders List**:

1. Select **Preferences…** from the **Edit** menu and click **Shielded Folders** in the IDE Preference Panels list. The CodeWarrior IDE displays the **Shielded Folders** configuration panel (see Figure 9-7 on page 9-16).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-7 Shielded Folders configuration panel

2. Enter the expression that specifies the directories you want to ignore in the Regular expression field. You can enter the name of a specific directory, for example `Proj_Backup`, or you can enter a regular expression that matches several directory names, for example, `.*[ _]Backup`.

3. Specify the type of operations to shield matching directories against. Enable the **Project operations** check box, the **Find and compare operations** check box, or both.

4. Click **Add** to add the regular expression to the **Shielded Folders** List. Bullets in the list indicate the operations for which the CodeWarrior IDE ignores matching directories.

5. Click **Apply** to apply your changes.

To modify an existing regular expression:

1. Select **Preferences…** from the **Edit** menu and click **Shielded Folders** in the IDE Preference Panels list. The CodeWarrior IDE displays the **Shielded Folders** configuration panel (Figure 9-7).

2. Select the regular expression you want to modify.

3. Modify the regular expression using the Regular expression text box.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Specify the operations you want to skip matching directories for, by enabling the **Project operations** check box, the **Find and compare operations** check box, or both.
5. Click the **Change** button to update the regular expression in the **Shielded Folders** List. Bullets in the list reflect the changes that you made to the corresponding operations.
6. Click **Apply** to apply your changes.

To remove an existing regular expression:

1. Select **Preferences…** from the **Edit** menu and click **Shielded Folders** in the IDE Preference Panels list. The CodeWarrior IDE displays the **Shielded Folders** configuration panel (see Figure 9-7 on page 9-16).
2. Select the regular expression you want to remove so that it appears in the Regular expression field.
3. Click the **Remove** button to delete the regular expression from the **Shielded Folders** List.
4. Click **Apply** to apply your changes.

9.3.6 Configuring global source trees

The CodeWarrior IDE predefines four global source tree types:

- absolute path
- project relative
- compiler relative
- system relative.

You can define additional global source trees in two panels:

Target Settings Panels, Source Trees

You can use the source trees you define in the Target Settings window with the current build target only. See *Configuring source trees* on page 10-29 for information on configuring target-specific source trees.

IDE Preference Panels, Source Trees

You can use the source trees you define in the IDE Preferences window with all projects. This section describes how to configure global source trees.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

If you define the same source tree in both panels, the target-specific source trees take precedence over the global source trees.

The IDE Preferences Source Trees configuration panel enables you to define global source trees (root paths) for use in your projects. Source trees are used in all dialog boxes that require you to select a path type. You can define your project access paths and build target output in terms of source trees. See *Configuring access paths* on page 10-12 for more information.

To add or change a source tree for all projects:

1. Select **Preferences…** from the **Edit** menu and click **Source Trees** in the IDE Preference Panels list to display the configuration panel (Figure 9-8). The Source Trees configuration panel displays a list of any source paths that are currently defined.

Figure 9-8 Source Trees configuration panel

2. Edit the source tree details:
   - To change an existing source path, click on the entry in the list of source trees. The source tree details are displayed in the relevant text fields.
   - To add a new source tree, type a name for the new source path in the Name field.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click the **Type** drop-down menu to select the type of source tree. Select one of:

   Absolute Path

   Select this option to specify a specific directory as the root for your source tree.

   Environment Variable

   Select this option to specify a directory defined in an environment variable as the root for your source tree.

   Registry Key

   Select this option to specify a directory defined in a Windows registry key as the root for your source tree.

4. Specify the source tree root:
   - If the source tree is an absolute path, click **Choose…** to select the root directory from the standard file dialog box.
   - If the source tree is defined in an environment variable, enter the name of the environment variable as the source tree root. If the environment variable you enter is defined, the CodeWarrior IDE adds the source tree to the list of defined source trees and displays the value of the environment variable.
   - If the source tree is a registry key enter the full path name of the registry key, without the prefix volume label (such as `My Computer`), and ending with the name of the registry entry. If the registry key is defined, the CodeWarrior IDE adds the source tree to the list of defined source trees and displays the value of the registry key.

5. Click one of:
   - **Add** if you are adding a new source tree.
   - **Change** if you are modifying an existing source tree.

6. Click **Apply** to apply your changes.

   The new source tree is displayed in all dialog boxes that require you to select a path type, for example, the Browse for directory dialog box used when adding access paths to CodeWarrior projects. (See *Configuring access paths* on page 10-12 for more information on adding access paths to CodeWarrior projects.)

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

If you have used a non-predefined global source tree as an access path in a project that you are providing to another developer, you must also provide a copy of the Source Trees configuration panel. Do this by using the **Export Panel...** button to save a copy of the configuration panel. See *Using the IDE Preferences window* on page 9-3 for information on using the **Export Panel...** button.

Removing a source tree

To remove a source tree:

1. Select **Preferences…** from the **Edit** menu and click `Source Trees` in the IDE Preference Panels list to display the Source Trees configuration panel (see Figure 9-8 on page 9-18).
2. Click the entry you want to modify in the Source Trees list.
3. Click **Remove** to remove the selected source tree.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.4 Choosing editor preferences

This section describes the preference panels that control editor features. The Editor panels are described in:

- *Code completion*
- *Code formatting* on page 9-22
- *Editor settings* on page 9-24
- *Font & Tabs* on page 9-29
- *Text Colors* on page 9-32.

9.4.1 Code completion

The Code Completion configuration panel enables you to define the behavior of the CodeWarrior IDE editor so that it automatically suggests ways to complete source code as it is entered in the editor window. For more information on code completion, see *Completing code* on page 6-15.

To configure code completion:

1. Select **Preferences…** from the **Edit** menu and click `Code Completion` in the IDE Preference Panels list to display the configuration panel (Figure 9-9).

Figure 9-9 Code Completion configuration panel

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click the options in the **Code Completion** group:

   Automatic Invocation

   Select this option to automatically open the Code Completion window to complete programming-language symbols. Clear this option to open the window manually.

   Display deprecated items

   Select this option to have the Code Completion window display deprecated items in gray text. Clear this option to have the window hide deprecated items.

   Window follows insertion point

   Select this option to have the Code Completion window follow the insertion point as you edit text. Clear this option to leave the window in place.

   Case sensitive

   Select this option to have the CodeWarrior IDE consider case when completing code. Clear this option to have the case ignored.

   Code Completion Delay (ticks)

   Edit this option to configure the amount of time to wait before the Code Completion window automatically opens after you stop typing in the editor window. Specify the delay in ticks, where a tick is 1/60 of a second.

3. Click **Apply** to apply your changes.

9.4.2 Code formatting

The Code Formatting configuration panel enables you to configure the automatic formatting style applied to code added in the editor window. For more information on Code formatting, see *Formatting code* on page 6-14.

To configure code formatting:

1. Select **Preferences…** from the **Edit** menu and click `Code Formatting` in the IDE Preference Panels list to display the Code Formatting configuration panel (Figure 9-10 on page 9-23).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-10 Code Formatting configuration panel

2. Click **Use Automatic Code Formatting** to enable automatic formatting of code.

3. Click on the **Language Settings** drop-down menu to select the language you want to configure:
   - C/C++
   - Java

   For each of these languages you can set various formatting options to apply to code written in this language. As you select an option, default selections change and the code in the **Sample** group box changes to show the formatting effect of each feature:

   Format braces "{}"

   Select this option to have the editor automatically insert a closing brace when you type an opening brace. The editor places the cursor between the opening brace which you type and the closing brace which it inserts.

   Place opening brace on separate line

   Select this option to have the editor place an opening brace that you type at the beginning of the next line.

   Indent braces

   Select this option to have the editor indent braces by one Tab stop from the previous line.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Place "else" on same line as closing brace

Select this option to have the editor place `else` and `else if` text on the same line as the closing brace of the `if` statement.

Indent code within braces

Select this option to have the editor indent code by one Tab stop from the braces.

Indent "case" within "switch" statements

Select this option to have the editor indent `case` statements by one Tab stop inside a `switch` statement.

Close braces, brackets, and parentheses

Select this option to have the editor automatically insert the corresponding closing character when you type an opening brace, bracket, or parenthesis. The editor places the cursor between the opening character and the closing character.

4. Click **Apply** to apply your changes.

9.4.3 Editor settings

This section describes how to configure the behavior of the CodeWarrior IDE editor using the Editor Settings configuration panel.

There are three groups of options. For information on changing the options see:

- *Setting Remember options*
- *Specifying Contextual Menus* on page 9-26
- *Specifying Other Settings* on page 9-27.

Setting Remember options

The Remember options determine the editor window settings that are saved from one programming session to the next. To set Remember options:

1. Select **Preferences…** from the **Edit** menu and click `Editor Settings` in the IDE Preference Panels list to display the Editor Settings configuration panel (see Figure 9-11 on page 9-25).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-11 Editor Settings configuration panel

2. Select the options you want the CodeWarrior IDE to remember between editing sessions:

Font preferences

Select this option to specify that font information for individual files is remembered. If this option is not selected, all files inherit the default font settings from the CodeWarrior IDE (see *Font & Tabs* on page 9-29 for more information).

Selection position

This option instructs the CodeWarrior IDE to remember what text was scrolled into view, and the location of the insertion point or selection, at the time the file is closed. You must turn this option off if you want the editor to go to the top of the file when it opens.

Window position and size

Select this option to save the position and size of editor windows when they are closed.

Note

The CodeWarrior IDE can remember the window position and selection position of a file only if the file is writable. Files might not be writable if you are using a version control system and have checked out a read-only copy of a file. (For more information about using the CodeWarrior IDE with a VCS, see the Metrowerks web site.)

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click **Apply** to apply your changes.

Specifying Contextual Menus

The Contextual Menus options toggle the display of specific types of menu commands in the contextual menus. To set Contextual Menus options:

1. Select **Preferences…** from the **Edit** menu and click `Editor Settings` in the IDE Preference Panels list to display the Editor Settings configuration panel (see Figure 9-11 on page 9-25).

2. Select the types of menu commands you want displayed in the contextual menus:

   Edit Commands

   Enable this check box to display commonly used menu commands from the **Edit** menu. See *Edit menu* on page D-5 for more information.

   Browser Commands

   Enable this check box to display commonly used menu commands from the **Browser** menu. See *Browser menu* on page D-13 for more information. After you enable this check box, you can also enable the **Insert Template Commands** check box to display templates from your source code. The contextual menus then display the **Insert** command, as shown in Figure 8-13 on page 8-24. If necessary, this command displays a submenu that lists the templates.

   Insert Template Commands

   Enable this check box to insert function templates into your source code.

   Project Commands

   Enable this check box to display commonly used menu commands from the **Project** menu. See *Project menu* on page D-10 for more information.

   Debugger Commands

   Enable this check box to display commonly used menu commands from the **Debug** menu.

   Note

   The **Debug** menu is not implemented by the ARM version of the CodeWarrior IDE.

   VCS Commands

   Enable this check box to display commonly used menu commands from the **Version Control** menu.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

The **Version Control** menu is *only* displayed if a version control system plug-in module has been installed (see *Overview of the editor window* on page 6-3).

3. Click **Apply** to apply your changes.

Specifying Other Settings

The Other Settings options control how the editor works. To change these settings:

1. Select **Preferences…** from the **Edit** menu and click `Editor Settings` in the IDE Preference Panels list to display the Editor Settings configuration panel (see Figure 9-11 on page 9-25).

2. Select the options you require:

Balance while typing

Select this option to instruct the CodeWarrior IDE to check for balanced parentheses, brackets, and braces as you type. When you type a right parenthesis, bracket, or brace, the editor attempts to locate the matching left counterpart. If the counterpart is found, the editor brings it into view, highlights it for a length of time specified by the Balance Flash Delay (which can be specified in the text box at the bottom of this pane), and returns to where you were typing. If the counterpart is not found, the editor beeps. By default, the **Balance while typing** option is on.

Note

If you want to check for balanced punctuation without highlighting it, set the Balance Flash Delay to 0.

Use multiple undo

Select this option to undo and redo multiple actions. If this option is not selected, you can undo or redo only the last action that you performed. See the menu options, **Redo**, **Multiple Undo**, and **Multiple Redo** in Table D-2 on page D-5 for more information.

Relaxed C popup parsing

Select this option if you use Kernighan and Ritchie (K&R style) coding conventions in your source code. This option instructs the CodeWarrior IDE to recognize and display function names in the **Functions** drop-down menu. You must deselect this option if you use nonstandard macros that can interfere with K&R-style code.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

Some macro functions are not recognized when this option is enabled. If you encounter problems with viewing function names, disable this option and try again.

Drag and drop editing

Select this option to enable drag-and-drop text editing support in the editor. See *Moving text with drag and drop* on page 6-13 for more information on drag and drop editor features.

Left margin click selects line

Select this option to enable left margin editing features. Moving the mouse pointer to the left edge of an editor window changes the mouse pointer into a right-pointing arrow. This enables you to:

- click the window to select the line at the mouse pointer
- click and drag the mouse to select more than one line.

When this option is not selected, the mouse pointer always faces left and cannot select an entire line with a click.

Sort function popup

Select this option if you want items in the **Functions** drop-down menu in the editor window to be sorted alphabetically by default. See *Using the Functions drop-down menu* on page 6-19 for more information.

Enable Virtual Space

Enable this check box to use virtual spaces in the editor. With virtual spacing enabled, you can use the arrow keys to move the text insertion point past the end of a line of source code. The editor automatically inserts spaces between the former end of the line and newly entered text.

Balance Flash Delay (ticks)

Use this text field to specify the amount of time the CodeWarrior IDE editor highlights an opening parentheses, bracket, or brace when balancing punctuation. Specify the delay in ticks, where a tick is 1/60 of a second. See *Balancing punctuation* on page 6-14, and the description of **Balance while typing** above for more information.

Enter a value of 0 (zero) if you want to disable flashing entirely.

Default file format

Use the **Default file format** drop-down menu to set the end-of-line conventions that the CodeWarrior IDE uses to create new files. You can choose from:

- DOS

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- UNIX
- Macintosh.

3. Click **Apply** to apply your changes.

9.4.4 Font & Tabs

The Font & Tabs configuration panel enables you to set the default font and tab information for the CodeWarrior IDE editor. You can change the:

- settings used for an individual file (see *Setting the font and tabs for a single file*)
- default settings used by the CodeWarrior IDE for all editor windows that do not have individual settings specified (see *Setting font and tabs defaults* on page 9-31).

Note

To change settings for individual files, you must ensure that the Remember option **Font preferences** is selected in the Editor Settings configuration panel.

Setting the font and tabs for a single file

To change the font settings for an individual file:

1. Ensure that the Remember option **Font preferences** is selected in the Editor Settings configuration panel. See *Setting Remember options* on page 9-24 for more information.
2. Ensure that the editor window you want to configure is the active window.
3. Select **Preferences…** from the **Edit** menu and click `Font & Tabs` in the IDE Preference Panels list to display the Font & Tabs configuration panel (Figure 9-12 on page 9-30).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-12 Font & Tabs configuration panel

4. Select the display font, font size, and script from the drop-down menus, if required. The font and size you select here are applied to the current editor window when you apply the settings.

5. Select Document tab settings if required:

   Auto Indent

   Select this option to maintain the current indent level when you press the Enter key.

   Tab Size Enter a tab size, in number of spaces. The CodeWarrior IDE:

   - sets the Tab character to the number of spaces you have selected, if the **Tab Inserts Spaces** option is not selected The size you select here is applied to the current editor window when you apply the settings.
   - sets the number of spaces inserted when the Tab key is pressed, if the **Tab Inserts Spaces** option is selected.

6. Select Global tab settings if required:

   Tab indents selection

   Select this option if you want the CodeWarrior IDE to indent selected lines when you press the Tab key. If this option is not set, selected lines are replaced with a Tab character when you press the Tab key.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

This option applies only to selected complete lines of text. If you select one or more words within a line and press Tab, the CodeWarrior IDE replaces the selection.

Tab Inserts Spaces

Select this option to insert space characters, instead of a Tab character, when you press the Tab key.

7. Click **Apply** to apply your changes. The CodeWarrior IDE applies your settings to the current editor file, and uses the same settings when you close and re-open the file.

Note

The CodeWarrior IDE can store the font settings for a file only if the file is writable. Files might not be writable if you are using a version control system and have checked out a read-only copy of a file. (For more information about using the CodeWarrior IDE with a VCS, see the Metrowerks web site.)

Setting font and tabs defaults

To set the default font and tab settings that the CodeWarrior IDE uses for all text files that do not have individual settings specified:

1. Ensure that the Remember option **Font preferences** is *not* selected in the Editor Settings configuration panel. See *Setting Remember options* on page 9-24 for more information.

   When this option is not selected, any changes you make in the Font & Tabs configuration panel apply to all CodeWarrior IDE editing sessions.

2. Follow the instructions in *Setting the font and tabs for a single file* on page 9-29 to select font and tab options.

   You do *not* require an open editor window to set default values.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.4.5 Text Colors

The Text Colors configuration panel enables you to change the default text coloring in your source code. It also provides four Custom Keyword sets that you can use to specify the text color for your own sets of keywords. A Custom Keywords set can contain function names, type names, or anything else you want highlighted in your editor windows.

To open the Text Colors configuration panel, select **Preferences…** from the **Edit** menu and click `Text Colors` in the IDE Preference Panels list (Figure 9-13).

Figure 9-13 Text Colors configuration panel

Note

Syntax Coloring options do not apply to assembly language source. However, you can use Browser Coloring to highlight assembly language constructs in the browser.

You can change:

- text foreground and background colors (see *Changing text colors* on page 9-33)
- specific syntax constructs (see *Changing syntax coloring* on page 9-33)
- browser colors (see *Changing browser coloring* on page 9-36).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Changing text colors

In the Text Colors group, set the following options:

Foreground This option configures the color of any text not affected by the options in the Syntax Coloring or Browser Coloring groups. Click on the color box to change the current color.

Background This option configures the color of the areas on which text appears. Click on the color box to change the current color.

Changing syntax coloring

The CodeWarrior IDE can use different colors for each type of text. Table 9-1 lists and describes the elements that can be configured.

Table 9-1 Syntax coloring highlights

Element	Description
Comments	Code comments. In C or C++, a comment is text enclosed by `/*` and `*/` or text from `//` to the end of the line.
Keywords	C and C++ language keywords. It does not include any macros, types, or variables that you or the system header files define.
Strings	Anything that is not a comment, keyword, or custom keyword. Sample strings include literal values, variable names, routine names, and type names.

Before you can change the syntax coloring options, you must enable the **Activate Syntax Coloring** check box.

Note

You can use **Syntax Coloring** from the **Document Settings** drop-down menu to turn syntax coloring on or off as you view a particular file. See *Overview of the editor window* on page 6-3 for more information.

Using color for custom keywords

You can use the Custom Keywords group of the Text Colors panel to select additional words to display in color. These words can be macros, types, or other names that you want to highlight. These keywords are global to the CodeWarrior IDE and will apply to every project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can also set target-specific colors for custom keywords. See *Custom Keywords* on page 10-82 for more information.

To add one or more keywords to a Custom Keyword Set:

1. Open the Text Colors configuration panel (see *Text Colors* on page 9-32).
2. Click **Edit…** to the right of the Custom Keyword Set you want to modify. The CodeWarrior IDE displays the Custom Keywords dialog box (Figure 9-14).

Figure 9-14 Custom Keywords dialog box

3. Type a keyword in the Custom Keywords text field. Check **Case Sensitive** if you want the CodeWarrior IDE to match the case of each keyword in the list when applying your color settings. You can also import sets of custom keywords that you have already saved. See *Importing and exporting custom keywords* on page 9-35 for more information.
4. Click **Add**. The CodeWarrior IDE adds the keyword to the Custom Keywords list.

   Note

   To delete a keyword from the list, select the keyword and then press Backspace. The CodeWarrior IDE removes the keyword from the Custom Keywords list.

   You might not be able to add keywords if the Custom Keywords list is very large. If the CodeWarrior IDE is unable to add a keyword to the list, it displays a dialog box informing you that adding the keyword was unsuccessful.
5. Click **Done** when you have finished. The dialog box is closed.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6. Click **Apply** to apply your changes. The changes you have made are applied to your current editor windows. All the custom keywords you have defined are displayed in the appropriate color.

Note

If you define a keyword in more than one Custom Keyword set, the CodeWarrior IDE uses the definition from the first set it encounters. For example, definitions in Custom Keyword Set 1 are used before those in Custom Keyword Set 2.

Importing and exporting custom keywords

You can use the Custom Keywords dialog box to export and import lists of defined keywords. To save a list of the keywords defined in a Custom Keyword Set, or to re-import a list of keywords that you have already saved:

1. Open the Text Colors configuration panel (see *Text Colors* on page 9-32).
2. Click **Edit** to the right of the Custom Keyword Set you want to export or import to. The CodeWarrior IDE displays the Custom Keywords dialog box (see Figure 9-14 on page 9-34).
3. Click either:
   - **Export to file…**, if you want to save the current list of keywords
   - **Import from file…**, if you want to load new keywords into the current list.

   The CodeWarrior IDE displays a standard file dialog box.
4. Depending on whether you are exporting or importing a keyword set, use the standard file dialog box to either:
   - enter the name of the file you want to save to
   - open the file that contains the list of keywords you want to import.

     If you are importing a custom keyword set, the CodeWarrior IDE adds the keywords in the imported file to any keywords already defined for the current set.
5. Click **Done** to close the dialog box and save your changes.
6. Click **Apply** in the Text Colors configuration panel to apply the changes to your current editor files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Changing browser coloring

The Browser Coloring section of the Text Colors configuration panel enables you to customize the browser. The browser can export its lists of symbols and their types to the CodeWarrior IDE editor. This enables the editor to use different colors for displaying various types of symbols.

Before you can change the browser coloring options, you must enable the **Activate Browser Coloring** check box.

The color choice for each symbol type is displayed in both the editor window and the browser window. Click the color box to change the current color.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.5 Setting commands and key bindings

This section describes how to:

- specify customized commands that can appear in the CodeWarrior IDE menus
- assign keyboard shortcuts to commands and change keyboard shortcuts that are already defined.

See also *Customizing toolbars* on page 9-48 for information on customizing CodeWarrior IDE toolbars.

9.5.1 Opening the Customize IDE Commands window

The Customize IDE Commands window contains two tabbed panels that enable you to specify your own commands, assign key bindings to commands, and customize the CodeWarrior IDE toolbar. To open the Customize IDE commands window:

1. Select **Commands & Key Bindings…** from the **Edit** menu. The CodeWarrior IDE displays the Customize IDE Commands window (Figure 9-15).

Figure 9-15 Customize IDE Commands window, Commands tab

2. Click either:
   - The **Commands** tab, to display the key bindings customization panel. The CodeWarrior IDE commands are displayed in a hierarchical list, ordered by their menu names.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- The **Toolbar Items** tab, to display the toolbar control elements that you can add to the CodeWarrior IDE toolbars. See *Customizing toolbars* on page 9-48 for more information.

3. Click a hierarchical control next to a command group to display the commands for the command group. Figure 9-16 shows an example for the Edit group of commands.

Figure 9-16 List of Edit commands

Note

Some commands are not implemented by the ARM version of the CodeWarrior IDE.

4. Click on a command to select it. The default key bindings for the command are displayed. See *Customizing key bindings* on page 9-42 for more information on modifying key bindings. See *Customizing toolbars* on page 9-48 for more information on adding commands to the CodeWarrior IDE toolbar.

9.5.2 Adding your own commands to the CodeWarrior IDE

You can use the **Commands & Key Bindings...** command in the **Edit** menu to add menu items and key bindings for your own external executable commands. You can add commands to existing CodeWarrior IDE menus, or you can create your own menu groups. To configure your own commands:

1. Open the Customize IDE Commands window. See *Opening the Customize IDE Commands window* on page 9-37.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click **New Group…** to create a new group if you want to create a new menu for the command. See *Creating a new command group* on page 9-40 for more information.

3. Select the group to which you want to add the command and click **New Command**.

Note

You cannot add your own commands to some default CodeWarrior IDE groups.

The CodeWarrior IDE adds an Action group to the Commands window (Figure 9-17).

Figure 9-17 Configuring a new command

4. Enter the name of the command to run in the Execute field, or click the **…** button and select the command path from the standard file dialog box.

5. Enter arguments to the command in the Arguments field, if required. You can click on the drop-down list button next to the text field to select arguments from a list of CodeWarrior IDE internal variables.

   For example, select the **Editor Selected Text** drop-down list item to specify the text selected in the current editor window as an argument to the command.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6. Enter the name of the working directory from which the command is to be executed, if required. You can click on the drop-down list button next to the text field to select arguments from a list of CodeWarrior IDE internal variables.

   For example, select the **Current Target Output File Directory** drop-down list item to specify the directory that contains the output from the currently selected build target.

   Figure 9-18 shows an example to specify a new **Mail** command in the Miscellaneous menu group.

Figure 9-18 Mail selection command

7. Define one or more key bindings for the new command, if required. See *Customizing key bindings* on page 9-42 for more information.

8. Click **Save** to save your settings. The CodeWarrior IDE adds the command to the specified menu group.

Creating a new command group

To create a new group for your own commands and optionally display it in the main menu:

1. Open the Customize IDE Commands window. See *Opening the Customize IDE Commands window* on page 9-37.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click **New Group…**. The CodeWarrior IDE inserts the new group into the commands list (Figure 9-19).

Figure 9-19 Creating a new group

3. Type a name for the new group in the Name text field.

4. Ensure that the **Appears in Menus** option is selected if you want to add a menu for the new group to the main CodeWarrior IDE menu bar.

5. Click **Save**. The CodeWarrior IDE changes the name of the new group and creates a new menu with the same name as the group.

Deleting a command or group

To delete a command or group that you have created:

1. Open the Customize IDE Commands window. See *Opening the Customize IDE Commands window* on page 9-37.

2. Select the command or group you want to delete.

3. Click **Delete**.

4. Click **Save**. The CodeWarrior IDE removes the selected command or group.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Note

Only remove commands or groups that you have created. Do not remove those that are predefined by the CodeWarrior IDE.

9.5.3 Customizing key bindings

This section describes how to customize the default CodeWarrior IDE key bindings definitions and options. You can:

- customize the keyboard shortcuts used for menu, keyboard, and editor commands in the CodeWarrior IDE
- attach or *bind* almost any key to any command, and define multiple key bindings for the same command
- set the key bindings for menu commands, source code editor actions, and other miscellaneous actions
- create multiple-keystroke command bindings.

This section describes:

- *Restrictions on choosing key bindings*
- *Adding a new key binding* on page 9-43
- *Deleting a key binding* on page 9-43
- *Using multiple-keystroke bindings* on page 9-44
- *Setting the Prefix Key Timeout* on page 9-44
- *Using a Quote Key prefix to create single-key key binding* on page 9-45
- *Setting Auto Repeat for key bindings* on page 9-46
- *Exporting key bindings* on page 9-47
- *Importing key bindings* on page 9-47.

Restrictions on choosing key bindings

The following restrictions apply to the keys you can bind to actions:

- Escape and Space keys are always invalid for key bindings.
- Function keys and the Clear key are valid for creating key bindings.
- Return and Tab keys require at least the Ctrl or Shift key. This restriction does not apply for the second key of a two-key sequence.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Adding a new key binding

To add a new key binding for a command:

1. Open the Customize IDE Commands window (see *Opening the Customize IDE Commands window* on page 9-37).
2. Click the hierarchical control next to the command group for the command you want to modify and select the command from the list.
3. Click **New Binding**. The CodeWarrior IDE displays the Edit Key Bindings dialog box (Figure 9-20).

Figure 9-20 Edit Key Bindings dialog box

4. Type the key sequence you want to use for the command, and click **OK** to confirm your setting. Click **Cancel** if you make a mistake.

   The new key binding is displayed in the Key Bindings field.
5. Click **Save** to save your settings.

Deleting a key binding

To delete a key binding:

1. Open the Customize IDE Commands window (see *Opening the Customize IDE Commands window* on page 9-37).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click the hierarchical control next to the command group for the command you want to modify and select the command from the list.

   Key bindings for the selected command are displayed in the Key Bindings field (see Figure 9-20 on page 9-43).

3. Select the key binding you want to remove and press the Delete key. The key binding is deleted from the list of key bindings.

4. Click **Save** to save your settings.

Using multiple-keystroke bindings

You can create multiple-keystroke command keys, such as those used in the Emacs text editor. For example, the key sequence in Emacs to save a file is Ctrl+X followed by Ctrl+S.

To emulate the Emacs key binding to save a file:

1. Open the Customize IDE Commands window and expand the Edit group and the File group (see *Opening the Customize IDE Commands window* on page 9-37).

2. Delete the Ctrl+X key binding in the Key Bindings field for the **Cut** command in the Edit group. You must delete the current key binding because you cannot assign the same key binding to more than one command.

3. Delete the Ctrl+S key binding in the Key Bindings field for the **Save** command in the File group.

4. Click **New Binding** and set the command key for the **Save** command to Ctrl+X Ctrl+S (see Figure 9-20 on page 9-43).

5. Click **Save** to save your settings.

You can adjust the maximum time to wait for a key press after a the first key sequence is pressed (see *Setting the Prefix Key Timeout*).

Setting the Prefix Key Timeout

The Prefix Key Timeout field sets the length of time that the CodeWarrior IDE waits for the second key sequence after the first sequence in a multiple-keystroke binding is pressed. Larger values indicate that the CodeWarrior IDE will wait longer for the second key sequence to be pressed.

To set the key timeout:

1. Locate the Key Bindings field for the multiple-keystroke binding you want to set (see *Opening the Customize IDE Commands window* on page 9-37).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Enter the required value for the timeout in the Prefix Key Timeout text field (see Figure 9-20 on page 9-43).

   Specify the timeout in ticks, where a tick is 1/60 of a second. Valid values are in the range 1 to 999. The default value is 120.

3. Click **Save** to save your settings.

Using a Quote Key prefix to create single-key key binding

Typically, a key binding requires you to use two keys in combination:

- a modifier key, such as the Ctrl key
- a printing key, such as the 1 key.

However, you can define key bindings that do not require a modifier key. For example, you can assign the number 1 key to a command with no modifier.

If you assign a key binding to a single printing key, you must then type a Quote Key prefix in order to ignore the key binding and type the printing character associated with the key. For example, if you have assigned the 1 key to a command and the tilde (~) key as the Quote Key, you must type ~1 in order to enter the character 1 into an editor window. To enter a tilde character you must type the tilde key twice.

Note

The Quote Key affects only the next key or combination of keys that you type. You must use the Quote Key once for each bound key or combination of keys for which you want to type the equivalent character.

By default, the CodeWarrior IDE does not define a Quote Key. To assign a Quote Key:

1. Open the Customize IDE Commands window (see *Opening the Customize IDE Commands window* on page 9-37).

2. Click the hierarchical control next to the Miscellaneous group and select the **Quote Key** entry (see Figure 9-21 on page 9-46).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-21 Selecting the quote key

3. Click **New Binding** to display the Edit Key Bindings dialog box (Figure 9-22).

Figure 9-22 Edit Key Bindings for the Quote Key

4. Type the key you want to use as the Quote Key prefix and click **OK** to set the key binding.

5. Click **Save** to save your settings.

Setting Auto Repeat for key bindings

You can use Auto Repeat to specify that a key binding is repeated automatically when you press and continue to hold down its key combination. For example, you can use Auto Repeat with the **Find Next** command to repeatedly find a search string in a file. If a command has more than one key binding, you cannot configure Auto Repeat separately for each key binding. The Auto Repeat setting applies to all the key bindings for the selected command.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Exporting key bindings

You can save your key bindings in a file so that you can import them into the CodeWarrior IDE at another time. To export your current key bindings:

1. Open the Customize IDE Commands window (see *Opening the Customize IDE Commands window* on page 9-37).
2. Click **Export...**. A standard file dialog box is displayed.
3. Select the location where you want to save the key bindings file, and click **Save**. The current key bindings are saved to the key bindings file.

   You can use your own file, for example, `Tutorial_project_keys.mkb`.

Importing key bindings

To import a key bindings set that you have previously saved:

1. Open the Key Bindings configuration panel (see *Opening the Customize IDE Commands window* on page 9-37).
2. Click **Import** and select the key bindings file from the standard file dialog box.
3. Click **Open** to read the key bindings from the file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.6 Customizing toolbars

This section describes how to customize the CodeWarrior IDE toolbars. It describes:

- *Toolbar overview*
- *Showing and hiding a toolbar* on page 9-49
- *Modifying a toolbar* on page 9-50.

9.6.1 Toolbar overview

A toolbar contains elements, represented by icons, which act as buttons. A toolbar can contain the following elements:

Commands These are buttons that execute CodeWarrior IDE menu commands when clicked.

Controls These are the CodeWarrior IDE interface controls such as **Document Settings**, **Functions**, **Header Files**, **Markers**, **Version Control**, and **Current Target** drop-down menus (see *Overview of the editor window* on page 6-3).

Figure 9-23 shows the default toolbar from the CodeWarrior IDE project window.

Figure 9-23 Project window toolbar

Toolbar types

The following types of toolbar are available in the CodeWarrior IDE:

- Project window toolbars. These are displayed at the top of project windows.
- Editor window toolbars. These are displayed at the top of editor windows, and in the editing pane of other windows, such as the message window.
- Browser window toolbars. These are displayed in the Class Browser window in the single-class and multi-class browser views.
- The main window toolbar.

When you modify a toolbar, the changes apply wherever toolbars of that type are displayed. For example, if you modify an editor window toolbar, the change affects all editor windows and editor panes. See *Modifying a toolbar* on page 9-50 for more information on adding and removing elements in toolbars.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Each toolbar type has a default configuration of elements that you can restore if you want to discard your changes. See *Restoring a toolbar to default settings* on page 9-52 for more information.

9.6.2 Showing and hiding a toolbar

The **View** menu contains a **Toolbars** submenu that enables you to show, hide, reset or clear a toolbar. There are separate menu items for the main window toolbar, and the toolbar for the currently active window. When you select a toolbar command it applies to all toolbars of the same type. Hiding a toolbar does not change the elements contained in the toolbar.

To show or hide a window toolbar:

1. Click on the window which you want to configure to make it the active window.
2. Select the **Toolbars** command from the **View** menu.
3. Select either:
   - **Hide Window Toolbar** from the **Toolbars** submenu to hide the toolbar
   - **Show Window Toolbar** from the **Toolbars** submenu to show the toolbar.

   You can also show or hide the editor window toolbar with the **Toolbar Disclosure** button. See *Displaying window controls* on page 6-7 for more information.

   Note

   When you hide an editor window toolbar, the default toolbar drop-down menu controls are displayed along the bottom of the editor window (Figure 9-24).

Figure 9-24 Default toolbar at bottom of editor window

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

9.6.3 Modifying a toolbar

You can modify a toolbar by:

- *Adding a toolbar element*
- *Removing a toolbar element* on page 9-52
- *Removing all toolbar elements* on page 9-52
- *Restoring a toolbar to default settings* on page 9-52.

There are restrictions on the elements you can add or remove from a toolbar. These are described in *Adding a toolbar element* and *Removing a toolbar element* on page 9-52.

If you modify a toolbar type, the changes apply to every instance of that toolbar type created after the modification. For example, if you customize the project window toolbar, the changes apply to every project window you open, not just the toolbar in the active project window. Windows that are already open are not affected.

Adding a toolbar element

To add an element to a toolbar:

1. Ensure that the destination toolbar to which you want to add the element is open.
2. Select **Commands & Key Bindings...** from the **Edit** menu. The Customize IDE Commands window is displayed. The window contains two tabs:

   Commands Use the **Commands** tab to add toolbar elements for menu commands.

   Toolbar Items Use the **Toolbar Items** tab to add interface elements, such as the dirty file pop-up menu, or the file path indicator.
3. Click on the tab for the type of toolbar element you want to add. If you are adding a command, click on the hierarchical control next to the command group to navigate to the command.
4. Click on the *icon* for the command or interface element you want to add, and drag it to the toolbar where you want to add the element (see Figure 9-25 on page 9-51).

   Note

   You must click on the icon. You cannot drag the element if you click on its name.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 9-25 Dragging a toolbar element

5. Release the mouse button to add the element to the toolbar.

If the toolbar accepts the element, framing corners are displayed in the toolbar. If you cannot add the selected element to this particular toolbar, framing corners are not displayed. There are several reasons why a toolbar will not accept an element:

- the toolbar is full
- the element already exists in the toolbar
- commands can be added to a window toolbar only for menu commands that are available when the current window is the active window
- the drop-down menus and the File path indicator in the **Toolbar Items** tab can only be added to the editor window toolbar.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Removing a toolbar element

To remove an element from a toolbar:

1. Ctrl+right-click on the element in the toolbar. The CodeWarrior IDE displays a context menu.
2. Select **Remove Toolbar Item** from the menu to remove the item.

   Note

   Some default toolbar items cannot be removed.

Removing all toolbar elements

To remove all elements from a toolbar:

1. Click on the window which you want to configure.
2. Ctrl+right-click on an element in the toolbar. The CodeWarrior IDE displays a context menu.
3. Select **Clear Toolbar** from the menu to remove the toolbar items.

Restoring a toolbar to default settings

To restore the default settings for a toolbar:

1. Click on the window which you want to configure.
2. Either:
   - Ctrl+right-click on an element in the toolbar.
   - Ctrl+right-click on the empty toolbar area.

   The CodeWarrior IDE displays a context menu.
3. Select **Reset Toolbar** from the menu to restore the default toolbar items.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Chapter 10
Configuring a Build Target

This chapter describes how to configure build target options, including Compiler, Assembler, and Linker options, for a specific build target in a project. Build target options specify how the CodeWarrior IDE should process a build target in a project. This chapter contains the following sections:

- *About configuring a build target* on page 10-2
- *Overview of the Target Settings window* on page 10-4
- *Configuring general build target options* on page 10-8
- *Using the Equivalent Command Line text box* on page 10-35
- *Configuring assembler and compiler language settings* on page 10-37
- *Configuring linker settings* on page 10-65
- *Configuring editor settings* on page 10-82
- *Configuring the debugger* on page 10-84.
- *Configuring miscellaneous features* on page 10-86.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.1 About configuring a build target

The Target Settings window controls settings that affect how the CodeWarrior IDE builds a specific build target within a project. The build settings that you specify in the Target Settings window apply to the currently selected build target only. This means that you must specify these settings for *each* build target in your project separately.

For example, if you are configuring a project based on the ARM® Executable Image project stationery and you want to change the PCS options for your project, you must configure the options for both the Release and Debug build targets. See *Setting the current build target* on page 3-42 for more information on selecting build targets.

The Target Settings window is organized into a series of panels that apply to a specific group of build options (see Figure 10-2 on page 10-6 for an example). The panels are divided into the following main groups:

Target
The panels in this group enable you to configure basic settings for the current build target, including the target name and the linker to use. The linker setting is particularly important because the CodeWarrior IDE uses the linker setting to determine which panels to display in the other groups.

Language Settings
The panels in this group apply specifically to the ARM toolchain. They enable you to set options for the RealView® Assembler and RealView Compiler, including PCS options, and debug and optimization options.

Note

Language settings are used to compile and assemble all source files within a single build target. You cannot specify individual settings for specific source files. You must use separate build targets to change the settings for one or more files.

Linker
The panels in this group enable you to configure the RealView `armlink` and `fromelf` tool when it is run as a post-linker.

Editor
This group contains one panel that enables you to configure custom keywords for the CodeWarrior IDE editor.

Debugger
The panels in this group enable you to select the debugger you want to use for debugging output, and for running executable images. These panels also enable you to configure options for the debugger you have selected.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Miscellaneous

This panel enables you to configure the disassembly features of the `fromelf` tool.

After you have configured the target, you can export the settings for each panel to XML. You can then import this settings file at any time.

10.1.1 Configuration recommendations

If you have based your CodeWarrior project on one of the ARM project stationery templates then many of the configuration options are preset to values that are likely to be appropriate for your project. For example, if your project is based on the ARM Executable Image project stationery, the project is configured to use:

- The RealView Compiler (`armcc`)
- The RealView Linker (`armlink`) to output executable ELF format images
- The RealView `fromelf` tool, to process the output from the linker
- The RealView Debugger, to debug and run executable image files.

However, there are a number of configuration panels that you should review in order to ensure that the configuration options are appropriate for your target hardware and development environment. In particular, you should review the:

- Target and PCS / Code Gen panels for the RealView Compiler and the Target and PCS panels for the RealView Assembler. Ensure that the settings are appropriate to your target hardware and procedure call standard preferences.
- RealView Linker panels, to ensure that the appropriate output is produced.
- RealView FromELF panels, if you are using `fromelf` to produce an image for embedding in ROM or disassembled source.

10.1.2 Creating project stationery

After you have configured the settings you require for each of the build targets in your project, you can create project stationery of your own so that you can create new projects based on your preferences. See *Creating your own project stationery* on page 4-29 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.2 Overview of the Target Settings window

This section gives an overview of how to use the Target Settings window to set build target options in the CodeWarrior IDE. It describes how to display the Target Settings window, and how to save and discard changes to the target settings. For detailed descriptions of how to set specific target options see:

- *Configuring general build target options* on page 10-8
- *Configuring assembler and compiler language settings* on page 10-37
- *Configuring linker settings* on page 10-65
- *Configuring editor settings* on page 10-82
- *Configuring the debugger* on page 10-84
- *Configuring miscellaneous features* on page 10-86.

Note

The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately.

You can use the **Import Panel...** and **Export Panel...** features to transfer configurations between projects. See *Applying or discarding changes* on page 10-6 for information on using this feature.

Many ARM tools can read settings from a file using the `--via` argument to share options. The Extras configuration panel enables you to specify a via file for each of the compiler, assembler, linker, and FromELF. See *RealView Compilation Tools Compiler User Guide* for more information on writing via files to supply command-line arguments and options to ARM development tools.

10.2.1 Using the Target Settings window

This section gives basic information on using the Target Settings window to configure Target options. When you change target settings, the changes you make apply to the currently selected build target in the current project.

Displaying Target Settings panels

To display Target Settings configuration panels for a specific build target in the current project:

1. Open the project file you want to configure. See *Opening a project* on page 3-13 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Set the current build target you want to configure as described in *Setting the current build target* on page 3-42. For example, select the build target from the **Build Target** drop-down menu (Figure 10-1).

Figure 10-1 Select build target

3. To display the Target Settings window for that build target, either:
   - click the **Target Settings** button in the project window.
   - select the **Settings** menu item for your build target from the **Edit** menu.

     The name of the **Settings** menu item matches the build target for your currently selected project. For example, if your current build target is Release, the **Settings** menu item is named **Release Settings…**.

   The CodeWarrior IDE displays a Target Settings window with a list of available Target Settings Panels on the left side of the window. The panel selected in the list is displayed on the right side of the window. Figure 10-2 on page 10-6 shows an example.

   Note

   The panels that are listed depend on the linker that is selected for the current build target. See *Configuring general build target options* on page 10-8 for more information on specifying a linker.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-2 Selecting a Target Settings panel

4. Select the panel you want to configure in the list. You can use the arrow keys or click the name of the panel.

   Each panel contains related options that you can set. The options you select apply to the currently selected build target in the active project.

5. Select the options you require. See the rest of this chapter for detailed descriptions of the options in each configuration panel.

Applying or discarding changes

Use the buttons in the Target Settings window to apply or discard your changes:

OK Click this button to save any changes that you have made and close the window.

Cancel Click this button to close the window without saving any changes you have made.

Apply Click this button to commit any changes you have made in any of the panels. If you have changed an option that requires you to recompile the project, the CodeWarrior IDE displays a confirmation dialog box. Click **OK** or **Cancel** depending on whether you want to keep your changes or not.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Factory Settings

Click this button to reset the current panel to the settings that the CodeWarrior IDE uses as defaults. Settings in other panels are not affected. Only the settings for the current panel are reset.

Note

Factory Settings defaults are *not* the same as ARM stationery defaults so use this button with care, particularly on ARM RealView panels such as the Compiler and Assembler.

Revert Click this button to reset the state of the current panel to its last-saved settings. This is useful if you start making changes to a panel and then decide not to use them.

Import Panel

Click this button if you want to import an XML settings file. Select the file you want to import and click the **Open** button to import the settings.

Export Panel

Click this button if you want to export the current settings to XML. Enter the name of the file you want to export the settings to and click the **Save** button. You can then import these settings at any time.

If you make changes in the Target Settings window and attempt to close it, the CodeWarrior IDE displays a settings confirmation dialog box (Figure 10-3).

Figure 10-3 Settings confirmation dialog box

Click:

- **Don't Save** to discard your changes and close the dialog box and the Target Settings window.
- **Save** to save your changes and close the dialog box and the Target Settings window.
- **Cancel** to close the dialog box and continue using the Target Settings window without saving changes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.3 Configuring general build target options

This section describes how to configure general options for a specific build target, such as the name of the output file, and the linker and post-linker:

- *Configuring target settings*
- *Configuring access paths* on page 10-12
- *Configuring build extras* on page 10-21
- *Configuring runtime settings* on page 10-24
- *Configuring file mappings* on page 10-25
- *Configuring source trees* on page 10-29
- *Configuring the RealView target* on page 10-32.

Note

The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately.

10.3.1 Configuring target settings

The Target Settings configuration panel enables you to specify basic settings for your project such as the:

- target name
- linker and post-linker to use
- output directory, and whether to use relative or absolute paths for project files.

Because the linker choice determines which panels are displayed in the Targets Settings Panels list, you must select the linker first before you can specify other target-specific options such as compiler and linker settings. Panels are added and removed from the Targets Settings Panels list as appropriate each time you select certain options.

The CodeWarrior IDE ensures that only the files affected by an option are marked for recompilation when you change the option.

To set the Target Settings for the selected build target within a project:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **Target Settings** in the Target Settings Panels list to display the configuration panel (Figure 10-4 on page 10-9).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-4 Target Settings configuration panel

3. Specify the settings for your target:

Target Name

Set the name of the current build target.

The name shown is the name you specified when you created the build target. This is not the name of your final output file. See *Output file naming conventions and locations* on page 4-36 for more information on how the CodeWarrior IDE names output files.

Linker Your choice of linker determines the compiler and assembler that the CodeWarrior IDE calls. Specify the linker from the drop-down menu:

None No link stage is required for the target. This means that source files are not compiled or assembled. You can use this option if you want to use the CodeWarrior IDE to maintain a collection of non-source files.

ARM RealView Linker

Use the ARM RealView Linker to link the output from the assembler and compiler. See *Configuring the RealView Linker* on page 10-66 for more information on linker options.

ARM RealView Librarian

Use the ARM RealView Librarian to create a library from the compiler and linker output.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

External Build Linker

Use this to create a CodeWarrior IDE project that makes use of an external command line driven build system, such as GNU `Make` or Windows command file (`.bat`).

If you select this option, an External Build configuration panel is added to the Linker group of panels. However, other panels are removed because they are not required when your project uses an external `Make` file.

See *Using an external Make file* on page C-6 for more information.

For the ARM RealView options, if you want the build target to create an executable image or an object library, then you must also make sure that the Output Type in the RealView Target panel is set to **Linker Output**.

Note

See *Configuring file mappings* on page 10-25 for more information about the file mappings associated with the linker. The file mappings determine which filename extensions the CodeWarrior IDE recognizes.

Pre-linker

This field is not used by the CodeWarrior IDE for RVDS.

Post-linker

Select a post-linker to process output from the linker:

None Do not use a post-linker. No post-link stage is required for the target. Use this option, for example, if you want to create object files only.

ARM RealView FromELF

Send the output from the linker to the ARM RealView `fromelf` utility. `fromelf` processes the output using the options set in the RealView FromELF configuration panel. Use this option to provide more information about the ELF images output by the linker or convert the ELF images to other formats. See *Configuring RealView fromelf* on page 10-76 for more information.

To create a binary output, the build target must create an executable image. Therefore, make sure that:

- the Linker is set to **ARM RealView Linker** in the Target Settings configuration panel.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- the output type is set to **Linker Output** in the RealView Target configuration panel.

When you select this option, a RealView FromELF configuration panel is added to the Linker group of panels.

Batch File Runner

Use this to run a DOS batch file as the final step in the link process. The Batch File Runner runs the first, and only the first, `.bat` file in the link order view of the project. See *Running batch files with the Batch File Runner* on page 4-48 for detailed information on using the Batch File Runner.

Note

Support for the Batch File Runner is deprecated and will be removed in a future release.

Output Directory

This field displays the location of the data directory containing your build output. By default, this is the same directory that contains your project file. For the ARM stationery, this is `{Project}`*`TargetName`*. Click **Choose…** to select a different directory.

If you specify a directory, the path is shown relative to the project directory. For example, `{Project}Debug` specifies the `Debug` directory below the project directory. If the specified directory does not exist when you open the project, it is created.

Save project entries using relative paths

Select this option to instruct the CodeWarrior IDE to store project entries as a relative path from one of the access paths.

The CodeWarrior IDE remembers the location even where it has to re-search for files in the access paths. If you use relative paths, you can easily move an entire project to another computer.

If this setting is not selected, project entries are stored by name. See the menu options, **Re-search for Files** and **Reset Project Entry Paths** in *Menu options from the Project menu* on page D-10 for more information.

Note

The standard CodeWarrior IDE uses this option to enable you to add multiple source files with the same name to your project. However, the CodeWarrior IDE for RVDS does not permit the adding of multiple copies of source files that produce output objects. See *Filename requirements* on page 3-26 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Click **Apply** to apply your changes without closing the Target Settings window.

10.3.2 Configuring access paths

You can use the Access Paths configuration panel to define directories that the CodeWarrior IDE searches for libraries, header files, and source files.

For more information on configuring access paths, see the rest of this section:

- *Default header file search paths* on page 10-13
- *Default header file search paths* on page 10-13
- *Setting access path options* on page 10-14
- *Adding an access path* on page 10-16
- *Changing or removing an access path* on page 10-19
- *Adding the default access path* on page 10-20
- *Host Flags* on page 10-21.

Types of access path

The CodeWarrior IDE defines two types of access path:

User Paths The RealView tools search these paths for:

- User header files. These are header files that you include with a `#include "name.h"` header file statement.
- User libraries. These are libraries that correspond to your `#include "name"` library statements.
- Your source files. When you add a source file to your project from any directory, the CodeWarrior IDE automatically adds the path for the source file to the User Paths list.

By default, the User Paths list contains `{Project}`. This is the directory that contains the open project.

System Paths

The RealView tools search these paths for standard header files.

When you create a project using ARM stationery, the System Paths list contains `{RVCT31INC}`, that is, the default directory for the ARM standard C++ library header files when RVDS is installed. (This system path refers to the same directory as the `$RVCT31INC` environment variable created when you install RVDS v3.1.)

For more information about environment variables and how they are used, see the *RealView Compilation Tools Compiler User Guide*.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Unless you are working with your own C libraries, it is not necessary to change these paths. If you want to specify a list of paths that are used to search for user libraries, you can add the linker option `--userlibpath` to the Equivalent Command Line text box as described in *Configuring the RealView Linker* on page 10-66.

By default, the System Paths list contains `{Compiler}`. This is the directory that contains the CodeWarrior IDE executable.

Default header file search paths

Be aware that:

- The RealView Compiler uses the search paths defined in the CodeWarrior IDE Access Paths configuration panel when it is called from the CodeWarrior IDE. This means that the default search paths are different from those used by the ARM compiler (`armcc`) when invoked from the command line. In particular, there is no *current place* (that is, Berkeley UNIX search rules are not followed).

  If you are using a default RVDS installation, these differences have no effect. If you have made changes to your default installation, such as editing system header files, the behavior of the command line tools and the CodeWarrior IDE is different.

  See the chapter describing how to use the compiler in *RealView Compilation Tools Compiler User Guide* for details of how the compiler searches for header files.

- You can speed up compiling and linking if you ensure that search paths defined in the CodeWarrior IDE are not searched recursively. See *Setting access path options* on page 10-14 for more information.

To change the default search paths, select the **Always Search User Paths** option (see Figure 10-5 on page 10-14) to search for system header files in the same way as user header files.

Note

Avoid adding include files to a CodeWarrior for RVDS project by using the `-I`, `-J`, `-fd` and `-fk` options in the Equivalent Command Line text box. If you do, CodeWarrior for RVDS has no knowledge of the included files and you lose the browser information and error processing associated with the included files. Instead, add the include file paths as described in *Adding an access path* on page 10-16.

If your standard header files are not in the default access search paths, the RVCT tools cannot find them when compiling, linking, or running your project. See *Adding an access path* on page 10-16 for information on adding access paths.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Setting access path options

There are a number of ways to change the way access paths are searched, including:

- specifying recursive searches
- turning off searches for specific access paths
- always searching user paths.

To set access path options for a build target:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (Figure 10-5).

Figure 10-5 Access Paths configuration panel

By default, the User Paths list contains the recursive path, `{Project}`. This is added to the search path for all projects created using the CodeWarrior IDE (see *Adding an access path* on page 10-16).

3. Select one of the following, depending on the path type you want to modify:
   - **User Paths**
   - **System Paths**.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

By default, the System Paths list contains the recursive path, `{Compiler}`. This is added to the search path for all projects created using the CodeWarrior IDE. When you create a project using ARM stationery, the System Paths list contains the Source Trees path `{RVCT31INC}` instead (see *Adding an access path* on page 10-16).

4. Specify the search options you want:

Specifying recursive searches

Note

It is strongly recommended that you do not use recursive searching in complex, or multi-user projects. See *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-38 for more information.

Click in the Recursive Search column next to a directory name to toggle searching recursively through the directory (Figure 10-6).

Figure 10-6 Access paths detail

Here:

- If a directory icon is displayed next to the name of the directory, the CodeWarrior IDE performs a recursive search on the path. That is, the CodeWarrior IDE searches that directory and all the directories within it.
- If a directory icon is not displayed, the CodeWarrior IDE searches the named directory only.

Note

You can speed up project compilation and linking by turning off recursive path searches and adding each specific path of every directory that contains your files to either the User Paths list or the System Paths list.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Disabling search for a directory

Click in the Search column next to a directory name to toggle searching that directory (see Figure 10-6 on page 10-15):

- if a check mark is displayed next to the name of a directory, the directory is searched.
- if a check mark is not displayed, the directory is not searched.

Note

You can prevent any directory and all its sub-directories in an access path from being searched by renaming the Windows directory with enclosing parentheses. For example, changing `GameImages` to `(GameImages)` excludes the directory from all subsequent searches. To add it to the search list, you must explicitly add it as an access path. Additionally, you can use any other wildcard or regular expression specified in the **Shielded Folders** panel in the **IDE Preferences** window. See *Configuring shielded folders* on page 9-14.

Always Search User Paths

Select this option to search for system header files in the same way as user header files. See *Default header file search paths* on page 10-13 for more information.

Note

It is strongly recommended that you select this option for complex, or multi-user projects. See *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-38 for more information.

If this option is not selected, and any source file is found in a system search path, it is effectively promoted to an unchanging system file. This means that if a later version of the source file is placed in a user search path, it is not found by the CodeWarrior IDE until this option is selected.

5. Click **Apply** to apply your changes without closing the Target Settings window.

Adding an access path

User access paths are added automatically by the CodeWarrior IDE when you add a source file to your project that is not in an existing access path. In addition, you can specify both User and System access paths explicitly. Access paths are searched in the order that they are defined in the User and System panels. See *Changing or removing an access path* on page 10-19 for information on changing the order of access paths.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

To add a new access path to a project:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (see Figure 10-5 on page 10-14).

3. Select one of the following, depending on the path type you want to add:
   - **User Paths**
   - **System Paths**.

4. Click **Add...**. The CodeWarrior IDE displays the **Browse for Folder** dialog box (Figure 10-7).

Figure 10-7 Browse for Folder dialog box

5. Select the path type you require from the **Path Type** drop-down menu.

   Note

   You can use relative paths to enable projects to contain two or more files with identical names. However, for large projects, using relative paths slows performance.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can select from the following path types:

Absolute Path

The CodeWarrior IDE defines the access path of the added directory relative to the root level of the startup volume, including all directories in between. You must update absolute access paths if you move the project to another system, rename the startup volume, or rename any of the directories along the access path.

Project Relative

The CodeWarrior IDE defines the access path of the added directory relative to the directory that contains the project. The directory is identified by the project recursive path, `{Project}`.

You do not have to update project relative access paths if you move a project, provided the hierarchy of the relative path is the same. You cannot create a project relative path to a directory on a volume other than the one on which your project file resides.

Compiler Relative

The CodeWarrior IDE defines the access path of the added directory relative to the following directory:

install_directory`\IDEs\CodeWarrior\...\win_32-pentium`

The directory is identified by the compiler recursive path, `{Compiler}`.

You do not have to update compiler relative access paths if you move a project, provided the hierarchy of the relative path is the same. You cannot create a compiler relative path to a directory on a volume other than the one on which your CodeWarrior IDE resides.

Note

This setting is not relevant to the RealView Compiler which is installed in a different directory.

System Relative

The CodeWarrior IDE defines the access path of the added directory relative to the base directory containing your operating system. The directory is identified by the compiler recursive path, `{System}`.

You do not have to update system relative access paths if you move a project, provided the hierarchy of the relative path is the same. You cannot create a system relative path to a directory on a volume other than the one on which your active operating system base directory resides.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Source Tree Relative

The **Path Type** drop-down menu contains the name of any source trees you have defined. If you select a source tree, the CodeWarrior IDE defines the access path of the added directory relative to a directory defined in either a build target source tree, or a global source tree. The directory is identified by the source tree recursive path, {`source_tree`}, where `source_tree` is the name of your defined source tree.

See *Configuring source trees* on page 10-29 for more information on defining source trees for a specific build target. See *Configuring global source trees* on page 9-17 for more information on defining source trees for all projects.

6. Select the directory that you want to add to the access path and click **OK** to add the path, or **Cancel** to leave the list unchanged.

Note

You can add a path by dragging and dropping the directory from the Windows Explorer window and onto the Access Paths list pane. The path is added as:

- a project-relative path if the directory is on the same volume as the CodeWarrior IDE
- an absolute path if it is on a different volume from the CodeWarrior IDE.

You can also drag and drop single paths between the Access Paths configuration panels for different build targets in the same project, and in other projects.

7. Click **Apply** to apply your changes without closing the Target Settings window.

Changing or removing an access path

To change an access path for a build target:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (see Figure 10-5 on page 10-14).
3. Select one of the following, depending on the path type you want to change or remove:
   - **User Paths**
   - **System Paths**.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select the path you want to change or remove and click either:

   Change...

   The CodeWarrior IDE displays the **Browse for Folder** dialog (see Figure 10-7 on page 10-17). Use this to navigate to the new directory location (see *Adding an access path* on page 10-16 for details).

   Remove The access path is removed from the list.

   Note

   You can use drag and drop to change the order of the User and System access path lists. Click the access path you want to re-order and drag it to its new location. Access paths are searched in the order in which they appear in the paths lists.

5. Click **Apply** to apply your changes without closing the Target Settings window.

Adding the default access path

To add the default paths to the User Paths and System Paths lists:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (see Figure 10-5 on page 10-14).

3. Select one of the following, depending on the path type you want to change or remove (see *Types of access path* on page 10-12):
   - **User Paths**
   - **System Paths**.

   See *Working with ARM project stationery* on page 10-21 for information on System Paths in the ARM-supplied project stationery.

4. Click the **Add Default** button to add the corresponding CodeWarrior IDE recursive path (see *Adding an access path* on page 10-16 for more information):
   - `{Project}` to the User Paths list, which identifies the top-level directory of the project (for Project Relative paths)
   - `{Compiler}` to the System Paths list, which identifies the following CodeWarrior directory (for Compiler Relative paths):

     install_directory`\IDEs\CodeWarrior\CodeWarrior\...\win_32-pentium`

   Note

   When you add the default path for a path type, the **Add Default** button is disabled for that path type. Therefore, only one default path can be added to each path type.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5. Click **Change...** to specify a path relative to the default path, if required.

6. Click **OK** to apply your changes and close the Target Settings window.

Working with ARM project stationery

When you create a project that uses ARM-supplied project stationery, the System Paths list contains `{RVCT31INC}` (see *Configuring access paths* on page 10-12). If you delete this by accident, or if you want to add it to an existing project, you cannot use the **Add Default** button. Instead, you must add the ARM-defined default path yourself using the **Add...** button (see *Adding an access path* on page 10-16 for details).

Host Flags

Click the **Host Flags** button to see a list of host platforms that can use an access path. This menu does not apply to the ARM version of the CodeWarrior IDE and so, by default, all host platforms are selected.

However, if you add a new access path, you should not deselect the **Windows** host flag, because this instructs the CodeWarrior IDE not to search the access path on a Windows-based host.

10.3.3 Configuring build extras

The Build Extras panel contains options that affect the way a project builds, including:

- how project information is cached
- whether browser information is generated
- whether a third-party debugger is used.

To modify the Build Extras for a build target:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **Build Extras** in the Target Settings Panels list to display the configuration panel (see Figure 10-8 on page 10-22).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-8 Build Extras configuration panel

3. Select values for the following options:

Use modification date caching

This option is disabled by default in the ARM stationery.

Select this option:

- to instruct the CodeWarrior IDE not to check the modification dates of files you have changed outside the CodeWarrior IDE
- if you edit files with the CodeWarrior IDE editor only, and are working in a single user environment.

Selecting this option reduces compilation time. However, if you have modified any sources outside the CodeWarrior IDE, these changes are not detected.

Deselect this option:

- if you have configured the CodeWarrior IDE to use a third-party editor
- you are working on a multi-user project with shared access to source files.

See *Configuring IDE extras* on page 9-9 for more information on using a third-party editor. See *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-38 for more information on using the CodeWarrior IDE in a complex build environment.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Cache sub-projects

This option is enabled by default in the ARM stationery.

Select this option to:

- improve multiproject updating and linking
- enable the CodeWarrior IDE browser to include browser information from target sub-projects. See Chapter 8 *Working with the Browser* for more information on the browser.

Deselecting this option reduces the amount of memory required by the CodeWarrior IDE.

Generate Browser Data From

Specify this if you want the CodeWarrior IDE to generate symbolic information the next time that your project is built.

Choose:

None If you do not want browser data to be generated.

Compiler

This option is not supported.

Language Parser

If you want the data to be generated from the language parser. Without this information, you cannot open browser windows for your project.

The language parser:

- updates browser data as you edit
- generates browser information describing types of global variables and class data (C/C++)

  Note

  It is not possible to generate browser data for Assembler sources used in your project.

- generates members and return types of global functions and class member functions, so that the code completion feature can list members accessed by . or -> on an object or function return type
- generates information for global function prototypes, for example, from system headers accessible and used by the code completion feature. Currently, they do not show up in browser contents window, although they are syntax-colored in the CodeWarrior IDE editor

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- generates `#include` dependencies at parse-time. New `#include` statements are dynamically added so that you do not have to compile to generate new browser data.

For more information on browser settings and options, see Chapter 8 *Working with the Browser*.

Note

The contextual menu features of the browser work in the CodeWarrior IDE editor, in addition to all browser windows. It is recommended that you generate browser data, even if you do not use the browser windows, so that you can use the context-sensitive browser menu features such as finding definitions, declarations, and multiple definitions in your source code, and using symbol name completion.

Use External debugger

Select this check box if you are using a third-party debugger instead of the RealView Debugger. You must specify the following options:

Application

Click **Browse...** to search for the non-RealView Debugger application you want to use. The application you select is automatically placed in the Application field.

Arguments

If you want the selected debugger to use arguments whenever it runs from within the CodeWarrior IDE, enter them in this field. See the documentation accompanying the debugger you are using for details on supported arguments.

Initial directory

Click **Browse...** to search for the working (initial) directory in which the debugger you have selected will run. The working directory should be the root directory of your project. This facilitates, for example, the process of searching for a file when you attempt to open it from the selected debugger.

4. Click **Apply** to apply your changes without closing the Target Settings window.

10.3.4 Configuring runtime settings

Not supported in CodeWarrior for RVDS.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.3.5 Configuring file mappings

Use the File Mappings configuration panel to associate a filename extension, such as `.c`, with a plug-in tool, such as the ARM RealView Compiler. The file mappings you define in this panel tell the CodeWarrior IDE which tool, if any, to use to process files with defined extensions. The default file mappings for the ARM version of the CodeWarrior IDE depend on the:

- project stationery you use to create a project
- currently selected build target.

Table 10-1 shows the default file mappings for ARM stationery projects.

Table 10-1 File mappings

Extension	File Type	Compiler	Description
`.c`	Text	ARM RealView Compiler	C source coded file. Compiled by the RealView Compiler in ARM state for projects using ARM project stationery or Thumb® state for projects using Thumb project stationery.
`.ac`	Text	ARM RealView Compiler	C source coded file.
`.tc`	Text	ARM RealView Compiler	Thumb C source file
`.cc`, `.c++`, `.cpp`, or `.cp`	Text	ARM RealView Compiler	C++ source coded file. Compiled by the RealView Compiler in ARM state for projects using ARM project stationery, or Thumb state for projects using Thumb project stationery.
`.acpp`	Text	ARM RealView Compiler	C++ source coded file
`.tcpp`	Text	ARM RealView Compiler	Thumb C++ source file
`.h`	Text	ARM RealView Compiler	C header file
`.hpp`	Text	ARM RealView Compiler	C++ header file
`.s`	Text	ARM RealView Assembler	Assembler source file
`.inc`	Text	ARM RealView Assembler	Assembler include file
`.o`		ARM ELF Importer	Object source file

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table 10-1 File mappings (continued)

Extension	File Type	Compiler	Description
`.a` or `.lib`		ARM ELF Importer	ARM library file
`.axf`		ARM ELF Importer	ARM image
`.sym`		ARM ELF Importer	A file containing symbol definitions
`.tdv`		ARM ELF Importer	Intermediate build file for a sub-project
`.txt`	Text	None	Text file
`.bat`	Text	None	DOS batch command file for post linker
`.scat`	Text	None	Scatter-loading description file

Note

- File mappings determine whether the CodeWarrior IDE recognizes files in the project. If you have trouble adding files to your project, or if the CodeWarrior IDE refuses a directory or file that is dragged and dropped on the project window, check the File Mappings configuration panel to ensure that the relevant file extensions are defined.
- File mappings sets are associated with the currently selected linker. If you change the selected linker, the defined list of file mappings also changes. See *Configuring target settings* on page 10-8 for more information.
- If you add any new file extensions that specify the ARM ELF Importer as the compiler, these are also treated as object files to be included in the link stage.

Adding or modifying a file mapping

To view and modify the file mappings for a build target:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel (Figure 10-9 on page 10-27).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-9 File Mappings configuration panel

The File Mappings list contains a File Type, associated Extension, and compiler choice for each filename extension in the list. This list tells the CodeWarrior IDE which tool, if any, to invoke for files with specified filename extensions.

Figure 10-9 shows the default filename extensions used by the CodeWarrior IDE for a project based on the ARM Executable Image project stationery. Projects based on Thumb stationery invoke the RealView Compiler in Thumb mode.

3. Click the entry you want to modify in the File Mappings list and enter the File Type for the file. Alternatively, click **Choose…** and select a file of the same type.

4. Enter the filename extension, such as `.c` or `.h`, for the chosen file type.

5. Click the **Flags** drop-down menu to set flags that determine how the CodeWarrior IDE treats files of the chosen type. If a flag is set, the File Mappings configuration panel displays a dot in the appropriate flag column (Figure 10-9). The flags are:

Resource File

This flag does not apply to the CodeWarrior for RVDS.

Launchable

Select this flag to instruct the CodeWarrior IDE to open this type of file with the application that created it when you double-click it in a Project window Files view.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Precompiled

Select this flag to instruct the CodeWarrior IDE to compile this type of file before other files. This option is useful for file that are used to generate files used by other source files or compilers. For example, this option enables you to create a compiler that translates a file into a C source code file and then compiles the C file. YACC (*Yet Another Compiler Compiler*) files are treated as precompiled files because YACC generates C source code to be compiled by a C compiler.

Note

You cannot develop CodeWarrior plug-ins using the CodeWarrior IDE for RVDS. However, you can use the standard Metrowerks CodeWarrior for Windows development environment to write your own plug-ins to work with the CodeWarrior IDE for RVDS.

Ignored by Make

Select this flag to instruct the CodeWarrior IDE to ignore files of this type when compiling or linking the project. This option is useful for documentation files that you want to include with your project.

6. Click the **Compiler** drop-down menu and select a compiler for the chosen file from the list of available tools. Select **None** if you want to be able to add files with the specified filename extension to your project, but you do not want to use a plug-in tool to process them.

7. Click the **Edit Language** drop-down menu to select the language you want to edit in. The CodeWarrior IDE applies the appropriate syntax coloring for the chosen language.

8. Click either:
   - **Add**, to add a new file mapping
   - **Change**, to change the configuration for a selected file mapping.

9. Click **Apply** to apply your changes without closing the Target Settings window.

Removing a file mapping

To remove a file mapping:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel (see Figure 10-9 on page 10-27).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Click the entry you want to modify in the File Mappings list.
4. Click **Remove** to remove the selected file mapping.

10.3.6 Configuring source trees

The Source Trees configuration panel enables you to define target-specific source trees (root paths) for use in your projects. You can define your project access paths and build target output in terms of source trees. If you do this, it makes it far easier to move your projects to another computer and also facilitates sharing of sources between developers.

You can define source trees in two panels:

Target Settings Panels, Source Trees

You can use the source trees you define in the Target Settings window with the current build target only. This section describes how to configure target-specific source trees.

IDE Preference panels, Source Trees

You can use the source trees you define in the IDE Preferences window with all projects and build targets created using the CodeWarrior IDE. See *Configuring global source trees* on page 9-17 for information on configuring global source trees.

Note

If you define the same source tree in both panels, the target-specific source trees take precedence over the global source trees.

When you create a project using one of the ARM stationery templates installed with RVDS v3.1, the environment variable `RVCT31INC` is specified in this panel. You can also specify a location as an absolute path or from a registry key. You do not have to include the `RVCT31LIB` environment variable.

To add or change a source tree for the current build target:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **Source Trees** in the Target Settings Panels list to display the configuration panel (see Figure 10-10 on page 10-30). The Source Trees configuration panel displays a list of currently defined source paths.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-10 Source Trees configuration panel

3. Edit the source tree details:
   - To change an existing source path, click on the entry in the list of source trees. The source tree details are displayed in the relevant text fields.
   - To add a new source tree, type a name for the new source path in the Name field.

4. Click the **Type** drop-down menu to select the type of source tree. Select one of:

   Absolute Path

   Select this option to specify a specific directory as the root for your source tree.

   Environment Variable

   Select this option to specify a directory defined in an environment variable as the root for your source tree.

   Registry Key

   Select this option to specify a directory defined in a Windows registry key as the root for your source tree.

5. Specify the source tree root:
   - If the source tree is an absolute path, click **Choose…** to select the root directory from the standard file dialog.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- If the source tree is an environment variable enter the name of the environment variable. If the environment variable is defined, the CodeWarrior IDE adds the source tree to the list of defined source trees and displays the value of the environment variable. (If the variable is not defined, an error is displayed in the Project Messages window.)
- If the source tree is a registry key enter the full path name of the registry key, without the prefix volume label (such as `My Computer`), and ending with the name of the registry entry. If the registry key is defined, the CodeWarrior IDE adds the source tree to the list of defined source trees and displays the value of the registry key. If the registry key is not defined, an error is displayed in the Project Messages window.

6. Click one of:
   - **Add** to add a new source tree.
   - **Change** if you are modifying an existing source tree.
7. Click **Apply** to apply your changes.

   The new source tree is displayed in all dialog boxes that require you to select a path type, for example, the **Browse for Folder** dialog box used when adding access paths to CodeWarrior projects as shown in Figure 10-11. (See *Configuring access paths* on page 10-12 for more information on adding access paths to CodeWarrior projects.)

Figure 10-11 Example Select Access path dialog box

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Removing a source tree

To remove a source tree:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **Source Trees** in the Target Settings Panels list to display the configuration panel (see Figure 10-10 on page 10-30).
3. Click the entry you want to modify in the Source Trees list.
4. Click **Remove** to remove the selected source tree.

10.3.7 Configuring the RealView target

The RealView Target configuration panel enables you to change the output name and output type for a project.

The default output filename is the same as the project name. For example, a project called `test` might produce the image file `test.axf`. You can use this panel to produce a different image file, for example, `new.axf`, without changing the name of the project. You might want to change the name of a project if the build target is to be a sub-target of another build target.

You can also change the output type, to enable you to use your output files with any non-ARM link tool that can process ELF object files.

To change the output settings for a project:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Target** in the Target Settings Panels list to display the configuration panel (see Figure 10-12 on page 10-33). The RealView Target configuration panel displays the current settings for the output name and output type.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-12 RealView Target configuration panel

3. If you want to produce output files with a different name to the project file, uncheck the **Use project name** option and enter the required name in the Name text box. The Name text box is grayed out when the **Use project name** option is checked.

4. Click the **Output Type** drop-down menu to select the output type:

Linker Output

Select this option if you are using the RealView Linker or Librarian. This is the default.

The output produced depends on the Linker setting in the Target Settings configuration panel:

- If the Linker is set to **ARM RealView Linker**, the output from this build target is either an executable image or a partially-linked object.
- If the Linker is set to **ARM RealView Librarian**, the output from this build target is an ARM object library.

See *Using the Target Settings window* on page 10-4 for more information.

Directory of objects

Select this option if you do not want to use the linker output.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

When this option is set, only object files are produced. This is the usual setting if the build target is to be a sub-target of another build target. The sub-target generates the object files, and the parent build target links the object files to create an executable image as in, for example, the ARM sub-target of the Thumb ARM Interworking Image stationery (see *Using the Thumb ARM interworking stationery* on page 4-7.)

This option also enables you to use your output files with any non-ARM link tool that can process ARM ELF object files.

5. The following check box provides more information about the output produced:

Echo ARM tool command-lines verbosely

Select this option to display the build commands in the Errors & Warnings window.

By default, build commands are not displayed in the Errors & Warnings window when you build a target. In this case, the window is displayed only if any errors or warning messages are issued, or if you have specified that remarks are to be displayed.

6. Click **Apply** to apply your changes without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.4 Using the Equivalent Command Line text box

The Equivalent Command Line text box is displayed at the bottom of configuration panels for the following groups of panels:

- Language Settings, RealView Assembler and RealView Compiler panels
- Linker, RealView Linker panel
- Target, RealView Target panel
- Debugger, RealView Debugger panel
- Miscellaneous, RealView Disassembly

Figure 10-13 shows an example of the RealView Compiler Target configuration panel.

Figure 10-13 Equivalent Command Line text box

The Equivalent Command Line text box:

- Displays the command-line equivalent for any options you select in the panel. For example, if you specify the Floating Point option `VFPv2`, the Equivalent Command Line text box changes to include `--fpu=VFPv2` on the command line.

- Enables you to edit the command line, and add command-line options that have no interface controls in the configuration panels. Not all command-line options to the ARM RealView Compiler, Assembler, Linker, and other tools, have equivalent interface controls in the configuration panels.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- Enables you to copy and paste language settings between build targets. This is useful because the **Import Panel...** and **Export Panel...** features do not allow you to copy the settings made in a single tab of a panel, or to add to existing settings.

Press the Enter key after you have edited the command line to apply the options.

Note

You can add new options to the end of the command line in the Equivalent Command Line text box. The command line is reordered when you press the Enter key.

When using the Equivalent Command Line text box, the following behavior applies:

- If you enter a command-line option with an equivalent panel interface setting, this setting is updated with the new value. For example, if you edit the command line to include `--fpu=VFPv3`, the Floating Point option `VFPv3` is selected.
- If you enter a command-line option with an equivalent panel interface setting on a RealView Compiler configuration panel that is not currently visible, this setting is also updated with the new value. For example, if you edit the command line to include `-O3`, the **3 (maximum optimization, restricted debug view)** check box is selected on the Debug / Opt configuration panel.
- If you enter a command-line option that is not recognized, it is highlighted in red and moved to the end of the command line.

For more information on the command-line options to the ARM RealView tools, see:

- *RealView Compilation Tools Compiler User Guide*
- *RealView Compilation Tools Compiler Reference Guide*
- *RealView Compilation Tools Assembler Guide*
- *RealView Compilation Tools Linker and Utilities Guide*.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.5 Configuring assembler and compiler language settings

This section describes the Language Settings group of panels:

- Use the RealView Assembler panel to configure options for the RealView Assembler, including:
  - the processor type or architecture version, and other capabilities of your target hardware
  - PCS options
  - debug options
  - listing options, diagnostics, and other miscellaneous options.

  In addition, you can use this panel to predefine variable values for the assembler.

  See *Configuring the RealView Assembler* on page 10-38 for details.

- Use the RealView Compiler panel to configure options for the RealView Compiler, including:
  - the processor type or architecture version, and other capabilities of your target hardware
  - PCS and code generation options
  - language mode, for example ANSI C or strict ANSI C
  - debug and optimization options
  - warnings and errors information, and other preferences.

  See *Configuring the RealView Compiler* on page 10-50 for details of all compiler configuration options.

Note

The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately. The settings for a build target are applied to all files in the build target. To speed up the configuration for multiple targets, you can use the Equivalent Command Line text box to copy settings from one build target to another.

For information on configuring the other tools in the ARM toolchain see:

- *Configuring linker settings* on page 10-65
- *Configuring the debugger* on page 10-84.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.5.1 Configuring the RealView Assembler

This section describes how to configure the RealView Assembler (`armasm`) from within the CodeWarrior IDE. It provides general descriptions of the available assembler options.

Note

Many of the configuration settings described here use default settings that are appropriate for most development projects. However, you should review at least the Target options and PCS options to ensure that they are suitable for your project.

The RealView Assembler panel consists of a number of tabbed configuration panels. The available options are described in:

- *Configuring the target and architecture*
- *Configuring PCS options* on page 10-42
- *Configuring assembler options* on page 10-44
- *Configuring predefined variables* on page 10-45
- *Configuring diagnostic information* on page 10-47
- *Reading assembler options from a file* on page 10-48.

These sections provide general descriptions of the available assembler options. For more detailed descriptions, see the *RealView Compilation Tools Assembler Guide*.

Configuring the target and architecture

Use the Target configuration panel to configure the target processor and architecture for the RealView Assembler:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Assembler** in the Target Settings Panels list and click the **Target** tab to display the configuration panel (see Figure 10-14 on page 10-39).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-14 RealView Assembler Target configuration panel

3. Select target options, as required:

Architecture or Processor

Select the processor or architecture for your target hardware from the drop-down menu. Some processor-specific instructions produce either errors or warnings if compiled for the wrong target architecture.

Note

If you select a processor from the **Architecture or Processor** drop-down menu, the following behavior applies:

- Specifying the processor selects the appropriate architecture and FPU.
- Some processor specifications imply an FPU selection. For example, when compiling in ARM state, choosing `ARM1136JF-S` implies VFPv2.

  Be aware that any *explicit* FPU (set with `--fpu`) on the command line overrides an *implicit* FPU.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select floating-point options:

Floating Point

Use this option to select the target *Floating Point Unit* (FPU) architecture. Floating-point instructions produce either errors or warnings if assembled for the wrong target FPU. Choose from:

[auto] Use the floating-point selection based on the chosen processor or architecture (using the **Architecture or Processor** drop-down menu).

This is the default setting.

VFPv3 Select this option to target hardware vector floating-point units conforming to architecture VFPv3.

VFPv3 is backwards compatible with VFPv2 except that VFPv3 cannot trap floating-point exceptions. If your project does not make use of floating-point exception trapping, it should work without any changes on VFPv3.

VFPv2 Select this option to target hardware vector floating-point units conforming to architecture VFPv2.

SoftVFP Select this option to use software floating-point linkage.

SoftVFP+VFPv2

Selects software floating-point library with software floating-point linkage, and requires a VFP unit conforming to architecture VFPv2. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

SoftVFP+VFPv3

Selects software floating-point library with software floating-point linkage, and requires a VFP unit conforming to architecture VFPv3. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

None Select this option if you are not targeting a hardware floating-point unit, nor using the software floating-point library. This produces an error if your code contains floats.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Floating Point Models

Use this option to select the floating-point model to use. The model specifies floating-point conformance and sets library attributes and floating-point optimizations. Choose from:

Full IEEE
All facilities, operations, and representations guaranteed by the IEEE standard are available in single and double-precision.

Fixed IEEE: round-to-nearest, no inexact exceptions
IEEE standard with round-to-nearest and no inexact exceptions.

Stateless IEEE: compatible with Java
IEEE standard with round-to-nearest and no exceptions. This mode is compatible with the Java floating-point arithmetic model.

C/C++ Language Standard
IEEE finite values with denormals flushed to zero, round-to-nearest, and no exceptions. This model is C and C++ compatible. This is the default option.

Fast mode, as supported by the VFP
Perform more aggressive floating-point optimizations that might cause a small loss of accuracy to provide a significant performance increase. This option results in behavior that is not fully ISO C and C++ standard-compliant, however, numerically robust floating-point programs behave correctly.

5. Select the instruction set to use:

Initial State

This option sets up the initial state of the RealView Assembler to assemble instructions in the source as ARM or Thumb instructions. This state remains until an `ARM`, `THUMB`, `CODE16`, or `CODE32` directive is processed in the code. Choose from:

[auto]
Uses the default state for the selected processor. This invokes the RealView Assembler with `armasm`, and is the default setting.

ARM
Start the assembler in ARM state. This instructs the assembler to interpret instructions as ARM instructions.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

This is equivalent to an `ARM` or a `CODE32` directive in the code, or to invoking the RealView Assembler with `armasm --arm` or `armasm --32`.

Thumb Start the assembler in Thumb state. This instructs the assembler to interpret instructions as 32-bit Thumb-2 instructions or 16-bit Thumb instructions using ARM syntax.

This is equivalent to a `THUMB` directive in the code, or to invoking the RealView Assembler with `armasm --thumb`.

Legacy Thumb

Start the assembler in Thumb state. This instructs the assembler to interpret instructions as 16-bit Thumb instructions using old Thumb syntax.

This is equivalent to a `CODE16` directive in the code, or to invoking the RealView Assembler with `armasm --16`.

6. Select the byte order used by your target hardware. This can be either **Little Endian** or **Big Endian**. The default setting is **Little Endian**.

7. Click **Apply** to apply your changes without closing the Target Settings window.

See the *RealView Compilation Tools Assembler Guide* for information on the ARM, Thumb, and Thumb-2 instruction sets.

Configuring PCS options

Selecting PCS (*Procedure Call Standard*) options sets the appropriate attribute values for the code sections generated by the assembler. The linker checks the attribute values at link time and generates error messages if you attempt to link incompatible objects.

Note

Selecting PCS options does *not* provide checks that your assembly language code conforms to a given PCS variant. You must ensure that your assembly language code follows the conventions required by the PCS options you select.

If you expect to call your assembly language code from C or C++, you must ensure that your RealView Compiler is configured to use the same calling standard. See *Configuring PCS and code generation options* on page 10-54 for details.

To configure the PCS settings for the RealView Assembler:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click **RealView Assembler** in the Target Settings Panels list and click the **PCS** tab to display the configuration panel (Figure 10-15).

Figure 10-15 RealView Assembler PCS configuration panel

3. Select the PCS variant that you require using the Procedure Call Standard Options group box:

ARM/Thumb interworking

Select this option if you are writing ARM code that you want to interwork with Thumb code, or Thumb code that you want to interwork with ARM code, and you are writing the correct interworking return instructions. The linker generates suitable interworking veneers if necessary at link time. See the description of interworking in the *RealView Compilation Tools Developer Guide* for more information.

Read-only position independent

Select this option to mark your code as read-only position-independent. When this flag is set, the assembler sets the PI attribute on read-only sections it outputs.

Read-write position independent

Select this option to mark your code as read-write position-independent. When this flag is set, the assembler sets the PI attribute on read-write sections it outputs.

4. Click **Apply** to apply your settings without closing the Target Settings window.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Configuring assembler options

To configure other assembler options, such as debug options and listing control:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Assembler** in the Target Settings Panels list and click the **Options** tab to display the configuration panel (Figure 10-16).

Figure 10-16 RealView Assembler Options configuration panel

3. Select debug options, as required:

   Source Line Debug

   Select this option to instruct the assembler to generate debug tables. This option is set by default for all ARM project stationery.

   Keep Symbols

   Select this option to keep local labels in the symbol table of the object file, for use by the debugger. This option is set by default for all ARM project stationery.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select options for code listings generated by the assembler, as required:

   Listing on
   Select this option to instruct the assembler to output a detailed listing of the assembly language it produces after resolving assembler constructs such as directives and pseudo-instructions. The listing is displayed in a new text window. You can define how the listing is displayed in this window in the Dimensions group box.

   Terse Deselect this option to display lines skipped due to conditional assembly in the listing.

   Cross-references
   Select this option to instruct the assembler to list cross-referencing information on symbols, including where they were defined and where they were used, both inside and outside macros.

5. Click **Apply** to apply your settings without closing the Target Settings window.

Configuring predefined variables

Use the Predefines configuration panel to configure the assembler to predefine a global variable and execute one of the `SET` directives to set its value, for example:

```
Debug SETL {TRUE}
```

The `SET` directives enable you to configure numeric, string, or logical variables. See the *RealView Compilation Tools Assembler Guide* for detailed information on using these directives.

To predefine a new variable:

1. Display the Target Settings configuration panel for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Assembler** in the Target Settings Panels list and click the **Predefines** tab to display the configuration panel (Figure 10-17 on page 10-46).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-17 RealView Assembler Predefines configuration panel

3. Enter the name of the variable in the Variable Name field. Alternatively, if you want to modify or delete an existing predefined variable, select the variable from the **List of Predefines** drop-down menu.

4. Select the directive you require. Use:
   - `SETA`, for numeric expressions
   - `SETL`, for logical expressions
   - `SETS`, for string expressions.

5. Enter the variable value using one of the following:
   - Enter the value in Numeric Value, if you have selected a `SETA` directive.
   - Click the appropriate Boolean, if you have selected a `SETL` directive.
   - Enter a quoted string in the String Value field, if you want use the `SETS` directive, for example, `"dollar string"`.

     When you add the variable, the CodeWarrior IDE automatically inserts the escape character prefix \ for the quotes, for example:

     ```
     --predefine="STRING_DEF SETS \"dollar string\""
     ```

     Note

     If you want to add this line to the Equivalent Command Line text box directly, you must include the escape characters. In this case, they are not inserted automatically.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6. Click either:
   - **Add**, if you are creating a new variable definition
   - **Replace**, if you are modifying an existing variable definition
   - **Delete**, to delete an existing variable definition.

7. Click **Apply** to apply your settings without closing the Target Settings window.

Configuring diagnostic information

To control the output of the diagnostic messages generated by the assembler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Assembler** in the Target Settings Panels list and click the **Diagnostics** tab to display the configuration panel (Figure 10-18).

Figure 10-18 RealView Assembler Diagnostics configuration panel

3. Select which diagnostic messages are to be suppressed or reclassified.

   Enter a tag, or a list of tags separated by commas, as required:

   Suppress Disables all diagnostic messages that have the specified tag(s).

   Set to warning

   Sets the diagnostic messages that have the specified tag(s) to Warning severity.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Set to error

Sets the diagnostic messages that have the specified tag(s) to Error severity.

No Warnings

Select this option to turn off all warning messages.

4. Click **Apply** to apply your settings without closing the Target Settings window.

Reading assembler options from a file

Use the Extras configuration panel to specify a via file for the assembler. A via file is a text file that contains additional command-line arguments to the assembler. Via files are read when the assembler is invoked. The via file options are processed after the options specified in the configuration panels, and override them if there is a conflict. You can use via files:

- to ensure that the same assembler settings are used by different build targets and projects
- to include exceptionally long command-line options to the assembler.

See the *RealView Compilation Tools Compiler User Guide* for a description of via file syntax.

To specify a via file:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Assembler** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 10-19 on page 10-49).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-19 RealView Assembler Extras configuration panel

3. Enter the path name of the via file, or click **Choose…** and select the via file from the standard file dialog.

4. Click **Apply** to apply your settings without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.5.2 Configuring the RealView Compiler

This section describes how to configure the RealView Compiler (`armcc`) from within the CodeWarrior IDE. It provides general descriptions of the available options used for compiling C and C++ source files.

Note

Many of the configuration settings described here use default settings that are appropriate for most development projects. However, you should review at least the Target options and PCS options to ensure that they are suitable for your project.

The RealView Compiler panel consists of a number of tabbed configuration panels. The available options are described in:

- *Configuring the target and architecture*
- *Configuring PCS and code generation options* on page 10-54
- *Configuring sources* on page 10-56
- *Configuring debug and optimization* on page 10-59
- *Configuring the preprocessor* on page 10-60
- *Configuring code generation* on page 10-56.
- *Configuring diagnostic information* on page 10-62
- *Reading compiler options from a file* on page 10-63.

These sections provide general descriptions of the available compiler options. For more detailed descriptions, see the *RealView Compilation Tools Compiler User Guide*.

Configuring the target and architecture

Use the Target configuration panel to configure the target processor and architecture for the RealView Compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Compiler** in the Target Settings Panels list and click the **Target** tab to display the configuration panel (Figure 10-20 on page 10-51).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-20 RealView Compiler Target configuration panel

3. Select target options, as required:

Architecture or Processor

Select the processor or architecture for your target hardware from the drop-down menu. Some processor-specific instructions produce either errors or warnings if compiled for the wrong target architecture.

Note

If you select a processor from the **Architecture or Processor** drop-down menu, the following behavior applies:

- Specifying the processor selects the appropriate architecture and FPU.
- Some processor specifications imply an FPU selection. For example, when compiling in ARM state, choosing `ARM1136JF-S` implies VFPv2.

  Be aware that any *explicit* FPU (set with `--fpu`) on the command line overrides an *implicit* FPU.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select floating-point options:

Floating Point

Use this option to select the target *Floating Point Unit* (FPU) architecture. Floating-point instructions produce either errors or warnings if assembled for the wrong target FPU. Choose from:

[auto] Use the floating-point selection based on the chosen processor or architecture (using the **Architecture or Processor** drop-down menu).

This is the default setting.

VFPv3 Select this option to target hardware vector floating-point units conforming to architecture VFPv3.

VFPv3 is backwards compatible with VFPv2 except that VFPv3 cannot trap floating-point exceptions. If your project does not make use of floating-point exception trapping, it should work without any changes on VFPv3.

VFPv2 Select this option to target hardware vector floating-point units conforming to architecture VFPv2.

SoftVFP Select this option to use software floating-point linkage.

SoftVFP+VFPv2

Selects software floating-point library with software floating-point linkage, and requires a VFP unit conforming to architecture VFPv2. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

SoftVFP+VFPv3

Selects software floating-point library with software floating-point linkage, and requires a VFP unit conforming to architecture VFPv3. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

None Select this option if you are not targeting a hardware floating-point unit, nor using the software floating-point library. This produces an error if your code contains floats.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Floating Point Models

Use this option to select the floating-point model to use. The model specifies floating-point conformance and sets library attributes and floating-point optimizations. Choose from:

Full IEEE
All facilities, operations, and representations guaranteed by the IEEE standard are available in single and double-precision.

Fixed IEEE: round-to-nearest, no inexact exceptions
IEEE standard with round-to-nearest and no inexact exceptions.

Stateless IEEE: compatible with Java
IEEE standard with round-to-nearest and no exceptions. This mode is compatible with the Java floating-point arithmetic model.

C/C++ Language Standard
IEEE finite values with denormals flushed to zero, round-to-nearest, and no exceptions. This model is C and C++ compatible. This is the default option.

Fast mode, as supported by the VFP
Perform more aggressive floating-point optimizations that might cause a small loss of accuracy to provide a significant performance increase. This option results in behavior that is not fully ISO C and C++ standard-compliant, however, numerically robust floating-point programs behave correctly.

5. Select the instruction set to use:

Initial State

Use this group to configure how the compiler handles C and C++ source files in a target. Choose from:

[auto] Invoke the RealView Compiler, using the default behavior, with `armcc`.

Compile all C and C++ files as ARM instructions but make adjustments to the startup configuration based on the filename extensions, for example, compile files with `.tc` and `.tcpp` filename extensions as Thumb instructions.

ARM Invoke the RealView Compiler with `armcc --arm`.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Select this option to override the file extension adjustments made by the RealView Compiler. This means that *all* C and C++ sources are compiled as ARM instructions regardless of the filename extension.

Thumb Invoke the RealView Compiler with `armcc --thumb`.

Select this option to override the file extension adjustments made by the RealView Compiler. This means that *all* C and C++ sources are compiled as Thumb instructions regardless of the filename extension.

6. Select the byte order used by your target hardware. This can be either **Little Endian** or **Big Endian**. The default setting is **Little Endian**.

7. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring PCS and code generation options

Selecting the PCS (*Procedure Call Standard*) options instructs the compiler to generate code that conforms to the appropriate PCS variant. In general, you must ensure that you use compatible calling standard options when you are compiling objects or libraries that you expect to link together.

In addition, if you are calling routines written in ARM assembly language you must ensure that your assembly language code follows the conventions required by the PCS options you select, and that the assembler is configured with the appropriate PCS options. See *Configuring PCS options* on page 10-42 for details.

To configure the PCS options for the ARM RealView Compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Compiler** in the Target Settings Panels list and click the **PCS / Code Gen** tab to display the configuration panel (see Figure 10-21 on page 10-55).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-21 RealView Compiler PCS / Code Gen configuration panel

3. Select the PCS variant options that you require using the Procedure Call Standard Options group box:

ARM/Thumb interworking

Select this option if you are writing ARM code that you want to interwork with Thumb code, or Thumb code that you want to interwork with ARM code. The linker generates suitable interworking veneers if necessary at link time. See the description of interworking in the *RealView Compilation Tools Developer Guide* for more information.

Note

ARM recommends that all code in mixed ARM/Thumb projects is compiled for interworking. At the very least, functions that may possibly be called from the other state should be compiled for interworking.

Read-only position independent

Select this option to mark your code as read-only position-independent. See the `ROPI` option in the *RealView Compilation Tools Compiler Reference Guide* for more information.

Read-write position independent

Select this option to mark your code as read-write position-independent. See the `RWPI` option in the *RealView Compilation Tools Compiler Reference Guide* for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring code generation

To configure code generation options for the RealView Compiler:

1. Select the **One ELF section per function** check box to generate one ELF section for each function in the source file. Output sections are named with the same name as the function that generates the section, but with an `i.` prefix. This option increases code size slightly (typically by a few percent) for some functions because it reduces the potential for sharing addresses, data, and string literals between functions.

2. Specify a linker feedback file or use the **Choose...** button to locate it. Linker feedback enables the efficient elimination of unused functions.

   If you specify a feedback file that does not exist, it is created by the link command and used in subsequent compilations.

3. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring sources

Selecting source options tells the RealView Compiler about the source language you are using and how it is to be compiled. See the *RealView Compilation Tools Compiler User Guide* for detailed information on C and C++ standards conformance and implementation details.

To configure source options for the ARM RealView Compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Compiler** in the Target Settings Panels list and click the **Source** tab to display the configuration panel (Figure 10-22 on page 10-57).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-22 RealView Compiler Source configuration panel

3. Select the language used by the compiler from the **Source Language** drop-down menu:

   [default]

   Select this option to compile C and C++ sources using the compiler defaults, that is:

   - ISO Standard C (1990), for all C code
   - ISO Standard C (1999), for all C code
   - ISO Standard C++ (1998), for all C++ code.

   See the *RealView Compilation Tools Compiler User Guide* for more information on language options.

4. Select additional source language options in the Other Language Settings group box:

   Strict ISO violations are issued as Warnings by default. Select this option to issue ISO violations as Errors.

   GNU Language Extensions

   Select this option to enable the GNU compiler extensions supported by the RealView Compiler.

   See the *RealView Compilation Tools Compiler Reference Guide* for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5. Select from the following C++ Language Settings for compiling C++ sources:

Inject friend in the normal class scope

Select this option to control whether or not the name of a class or function that is declared only in `friend` declarations is visible when using the normal lookup mechanisms.

When `friend` names are injected, they are visible to these lookups. When `friend` names are not injected (as required by the standard), function names are visible only when using argument-dependent lookup and class names are never visible.

Do not parse templates in their generic form

Select this option to disable the parsing of non-class templates in their generic form, that is, even if they are not instantiated. If dependent name processing is enabled, templates are parsed by default.

Do not look-up template names when parsing

Select this option to disable dependent name processing, that is, the separate look-up of names in templates at the time the template is parsed, and at the time it is instantiated.

Interpret 'operator new' as 'operator nothrow new'

Select this option if you want all new `operator new` calls in a compilation to be treated as `operator new nothrow`.

Implicit use of std namespace

Select this option to enable the implicit use of the `std` namespace option when standard header files are included.

No implicit includes

Select this option to disable implicit inclusion of source files as a method of finding definitions of template entities to be instantiated.

Enable C++ exceptions

Select this option to configure the compiler to perform unwinding of functions at runtime. Function unwinding is implemented by emitting an exception table describing the operations to be performed.

Note

For projects with mixed C and C++ sources, any C++ Language Settings are ignored when compiling the C sources.

6. Click **Apply** to apply your changes without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Configuring debug and optimization

Use the Debug / Opt configuration panel to set debug controls, and optimization levels and criteria for the compiler. The optimization selections you make affect the quality of the debug view of your code.

To configure optimization and debug options for the compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Compiler** in the Target Settings Panels list and click the **Debug / Opt** tab to display the configuration panel (Figure 10-23).

Figure 10-23 RealView Compiler Debug / Opt configuration panel

3. Select debug control options:

   Enable debug table generation

   Select this option to instruct the compiler to generate debug information. This option enables you to debug your output images at the source level. If this option is not selected, source level debugging is not available for the image.

   Improve debug view by not inlining

   Select this option to instruct the compiler to compile `inline` functions out of line so that they can be debugged at source more easily.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select the level of optimization:

   0 (minimum optimization, best debug view)

   Select this option to disable most compiler optimizations. Use this option in combination with enabled debug table generation to generate the best possible debug view of your output image and the lowest level of optimization.

   1 (restricted optimization, good debug view)

   Select this option to disable compiler optimizations that impact seriously on the debug view. Use this option in combination with enabled debug table generation to generate code that provides a good compromise between optimization and debug.

   2 (high optimization, restricted debug view)

   Select this option to enable high optimization. If this option is used in combination with enabled debug table generation, the debug view might be less satisfactory because the mapping of object code to source code is not always clear.

   3 (maximum optimization, restricted debug view)

   Select this option to enable all compiler optimizations. This option might result in a restricted debug view of your output image due to code movement and register re-use.

   If you select the option to generate debug information but do not specify the level of optimization, the ARM compiler assumes a default of `2`, that is, it adds `-O2` to the command line. When you build your project, the CodeWarrior IDE displays a warning message in the message window.

5. Select the optimization criterion:

   Space Select this option to favor small code size over execution speed. This is the default.

   Time Select this option to favor execution speed over code size.

6. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring the preprocessor

Use the Preprocessor configuration panel to configure preprocessor macros and to set search path options. To add, replace, or delete a preprocessor macro for the RealView Compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click **RealView Compiler** in the Target Settings Panels list and click the **Preprocessor** tab to display the configuration panel (Figure 10-24).

Figure 10-24 RealView Compiler Preprocessor configuration panel

3. If you are modifying an existing definition, double-click on the entry in the list of `#DEFINE` macro definitions so that it appears in the edit box. (You can create a new macro, by typing over the existing macro name.)

   If you want to add a new definition, enter the new macro name in the edit field, for example, `EXAMPLE_DEFINE`.

4. Edit the value of the macro definition as appropriate, for example:

   ```
   EXAMPLE_DEFINE=2
   ```

   If you do not enter a value, the value defaults to `1`.

5. Click either:
   - **Add** to add a new macro definition to the list
   - **Replace** to edit the value of an existing macro definition.

6. If you want to delete an existing definition, double-click on the entry in the list of `#DEFINE` macro definitions so that it appears in the edit box. Then click **Delete**.

7. Click **Apply** to apply your changes without closing the Target Settings window.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Configuring diagnostic information

The RealView Compiler issues messages warning about portability problems or other potential problems in your code. You can use the **Diagnostics** tab to configure the compiler to suppress or enable specific warnings.

To control the output of the diagnostic messages generated by the compiler:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Compiler** in the Target Settings Panels list and click the **Diagnostics** tab to display the configuration panel (Figure 10-25).

Figure 10-25 RealView Compiler Diagnostics configuration panel

3. Select which diagnostic messages are to be suppressed or reclassified.

   Enter a tag, or a list of tags separated by commas, as required:

   Suppress Disables all diagnostic messages that have the specified tag(s).

   Set to warning

   Sets the diagnostic messages that have the specified tag(s) to Warning severity.

   Set to error

   Sets the diagnostic messages that have the specified tag(s) to Error severity.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Select warning options that apply to both C and C++, as required:

   No warnings
   Select this option to turn off all warnings.

   Warn for strict ansi
   Select this option to enable warning messages issued when extensions to the ANSI standard are used implicitly, for example:

   - using an unwidened type in an ANSI C assignment
   - specifying bitfields with a type of **`char`**, **`short`**, **`long`**, or **`long long`**
   - specifying **`char`**, **`short`**, **`float`**, or **`enum`**, arguments to variadic functions such as `va_start()`.

   Remarks Select this option if you want the compiler to issue remark messages, such as warning of padding in structures. These messages are not issued by default.

5. Click **Apply** to apply your changes without closing the Target Settings window.

Reading compiler options from a file

Use the Extras configuration panel to specify a via file for the compiler. A via file is a text file that contains additional command-line arguments to the compiler. Via files are read when the compiler is invoked. The via file options are processed after the options specified in the configuration panels, and override them if there is a conflict. You can use via files:

- to ensure that the same compiler settings are used by different build targets and projects
- to include exceptionally long command-line options to the compiler.

See the *RealView Compilation Tools Compiler User Guide* for a description of via file syntax.

To specify a via file:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Compiler** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 10-26 on page 10-64).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-26 RealView Compiler Extras configuration panel

3. Enter the path name of the via file, or click **Choose…** and select the via file from the standard file dialog box.

4. Click **Apply** to apply your changes without closing the Target Settings window.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.6 Configuring linker settings

This section describes how to configure the RealView Linker from within the CodeWarrior IDE. The linker settings you can configure depend on which linker is selected in the Target Settings window. If you want to process your output with more than one linker or post-linker you can use dependent sub-projects or sub-targets to select additional linkers or post-linkers. See *Working with multiple build targets and sub-projects* on page 3-40 for more information.

You can use the following linker and post-linker options with the ARM toolchain:

RealView Librarian

There is no configuration panel for the ARM RealView Librarian (`armar`). You can use the Librarian to combine ELF object files into a library (see *Creating a project that builds an object library and ROM images* on page 4-14 for more information).

For more information on `armar`, see the description of creating and using libraries in the *RealView Compilation Tools Linker and Utilities Guide*.

RealView Linker

Use the panel to configure the RealView Linker. See *Configuring the RealView Linker* on page 10-66 for details.

RealView FromELF

The **RealView FromElf** panel is added to the Linker group when you:

1. Select **RealView Target** in the Target Settings Panels list.
2. Set Linker to **ARM RealView Linker**.
3. Set Post-linker to **ARM RealView FromELF**.

Use this to configure the `fromelf` utility as a post-linker to process output from the linker. The `fromelf` utility can perform a number of format conversions on linker output, such as converting an ELF image file to plain binary format suitable for embedding in ROM. See *Configuring RealView fromelf* on page 10-76 for details.

The Batch File Runner

There is no configuration panel required for the Batch File Runner. See *Running batch files with the Batch File Runner* on page 4-48 for more information. The Batch File Runner is deprecated in this release.

Note

The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.6.1 Configuring the RealView Linker

This section describes how to configure the RealView Linker (`armlink`) from within the CodeWarrior IDE. It provides general descriptions of the available linker options.

The RealView Linker panel consists of a number of tabbed configuration panels. The available options are described in:

- *Configuring linker output*
- *Configuring linker options* on page 10-69
- *Configuring image layout* on page 10-70
- *Configuring linker listings* on page 10-72
- *Configuring linker diagnostics* on page 10-74
- *Configuring linker extras* on page 10-75.

These sections provide general descriptions of the available linker options. See the *RealView Compilation Tools Linker and Utilities Guide* for more detailed descriptions of:

- the `armlink` command-line options
- how the linker constructs images and partially linked objects
- how to use scatter-loading description files to produce complex images.

Configuring linker output

Use the linker output panel to configure basic linker options that determine the type of image it produces. You can configure the linker to produce three basic types of output file. The options available in this panel depend on the type of image you select. You can choose to produce:

- A partially linked ELF object file that you can use in a later linker or Librarian operation. See also *Working with multiple build targets and sub-projects* on page 3-40 for information on configuring dependent sub-targets and sub-projects.
- A simple image that does not require a scatter-loading description file to specify the structure of the image. This provides an easy way to produce an executable ELF image but gives limited control over the structure of the image.
- A scatter-loaded image that gives more detailed control over the linker output. You must write a scatter-loading description file for the image you want to produce.

To set the output options:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

2. Click **RealView Linker** in the Target Settings Panels list and click the **Output** tab to display the configuration panel (Figure 10-27).

Figure 10-27 RealView Linker Output configuration panel

3. Select the type of output:

Partial Select this option to produce a partially linked ELF object file.

Simple Select this option to produce a simple ELF image file without using a scatter-loading description file. If you select this option the Simple image group of options becomes active and a **Layout** tab is added to the list of available configuration panels.

Also see *Configuring image layout* on page 10-70 for more information on specifying layout options when linking a simple image.

Scattered Select this option if you want to use a scatter-loading description file to specify the linker output. If you select this option, the Scatter description text field becomes active. You must provide a scatter-load description file to be used by the linker. Either:

- enter the path name of the file in the **Scatter description file** text field
- click **Choose…** to select the file from the standard file dialog box.

By convention, the CodeWarrior IDE expects scatter-load description files to use a filename extension of `.scat`. You can add scatter-load description files to your project.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. If you have selected a Simple image type in the previous step, the Simple image output options become available. These options give you limited control over the type of image output by the linker. The following options are available:

   RO Base Sets both the load and execution addresses of the region containing the RO output section at the address specified. The address must be word-aligned. If this option is not specified, and no scatter-loading description file is specified, the default RO base address is `0x8000`.

   RW Base Sets the execution addresses of the region containing the RW output section at the address specified. The address must be word-aligned.

   Ropi Select this option to instruct the linker to make the execution region containing the RO output section position-independent.

   Rwpi Select this option to instruct the linker to make the execution region containing the RW and ZI output sections position-independent.

   Split Image

   Splits the default load region, containing the RO and RW output sections, into the following load regions:

   - One root region containing the RO output section. The default load address is `0x8000`, but a different address can be specified with the RO Base option.
   - One root region containing the RW and ZI output sections. The load address is specified with the RW Base option. This option requires a value for RW Base. If RW Base is not specified, a base address of `0` is assumed.

   Relocatable

   Select this option to create a relocatable ELF image. Selecting this option is equivalent to producing a scatter file with `RELOC` defined as an attribute of every region.

5. Select a file containing the global symbol definitions from the image, if required. See the description of the `--symdefs` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

6. Select a steering file containing commands to edit the symbol tables, if required. See the description of the `--edit` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

7. Select a feedback file, if required, to inform the compiler about unused functions. See the description of the `--feedback` option in the *RealView Compilation Tools Compiler Reference Guide* for more information.

8. Click **Apply** to apply your changes without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Configuring linker options

Use the Options panel to configure options such as unused section elimination and symbol resolution:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Linker** in the Target Settings Panels list and click the **Options** tab to display the configuration panel (Figure 10-28).

Figure 10-28 RealView Linker Options configuration panel

3. Select options, as required:

   Remove unused sections

   Select this option to remove unused sections from the image.

   Caution

   You must take care not to remove interrupt handlers when you remove unused (read-only) sections. See the *RealView Compilation Tools Linker and Utilities Guide* for more information on unused section elimination.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

This retains all input sections in the final image even if they are unused.

Include debugging information

Select this option to instruct the linker to include debug table information from the output image. The image output by the linker is larger, but you can debug it at source. This is the default setting.

If this option is not selected, the linker discards any debug input sections it finds in the input objects and library members, and does not include the symbol and string table in the image.

If you are creating a partially linked object rather than an image, the linker discards the debug input sections it finds in the input objects, but does produce the symbol and string table in the partially linked object.

Search standard libraries

Select this option to instruct the linker to scan libraries to resolve references.

Give progress information while linking

Select this option to print progress information during linking.

Report "might fail" conditions as errors

Select this option to instruct the linker to report conditions that might result in failure as Errors (rather than Warnings).

Output local symbols

Select this option to instruct the linker to add local symbols to the output symbol table when producing an executable image.

4. Enter a value for the image entry point, if required, as:
   - a unique entry point address, for example:
     `--entry 0x0`
   - the start of the input section that defines a symbol, for example:
     `--entry reset_handler`
   - an offset inside a section within a specific object, for example:
     `--entry 8+startup.o(startupseg)`

   The entry point specified here is used to set the ELF image entry address. See the description of the `--entry` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

5. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring image layout

You can use the Layout configuration panel to specify which sections should be placed first and last in an image.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can use these panel settings, for example, to place:

- the section containing the reset and interrupt vector addresses first in an image
- an input section containing a checksum last in the RW section.

Note

You can use a scatter-loading description file to specify more complex ordering of sections. See the *RealView Compilation Tools Linker and Utilities Guide* for more information.

To configure the layout of ELF images output by the linker:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Linker** in the Target Settings Panels list and click the **Layout** tab to display the configuration panel (Figure 10-29).

Figure 10-29 RealView Linker Layout configuration panel

3. Enter section specifications in the appropriate fields, as required. You can enter specifiers for either, or both, the first and last sections:

   Place at beginning of image

   Select this option to place an input section first in its execution region. See the description of the `--first` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Place at end of image

Select this option to place an input section last in its execution region.

See the description of the `--last` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

Section specifiers can be:

- A symbol. This selects the section that defines the symbol. You must not specify a symbol that has more than one definition, for example:
  `--first reset`
- An object name, and section label. This selects the specified section from within the specified object, for example:
  `--first init.o(init)`

4. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring linker listings

Use the Listings panel to instruct the linker to output link information to a listing file:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Linker** in the Target Settings Panels list and click the **Listings** tab to display the configuration panel (Figure 10-30).

Figure 10-30 RealView Linker Listings configuration panel

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Select the information you want to list during the link operation:

   Image map
   Select this option to create an image map listing the address and the size of each load region, execution region, and input section in the image, including linker-generated input sections.

   Symbols Select this option to list each symbol used in the link step, including linker-generated symbols, and its value. This does not include mapping symbols.

   Mangled C++
   Select this option to instruct the linker to display mangled C++ symbol names in diagnostic messages, and in listings produced by the other Listings panel options. If this option is selected, the linker does not unmangle C++ symbol names. The symbol names are displayed as they appear in the object symbol tables.

   Section cross-references
   Select this option to list all cross-references between input sections.

   See the *RealView Compilation Tools Linker and Utilities Guide* for more information on these command-line options.

4. Enter the name to be used for the list file, or click **Choose…** to select a listing file from the standard file dialog box. If you do not enter a filename the listing information is displayed in a message window.

5. Enable the **Static Callgraph** check box if you want to generate a static callgraph of functions. This option generates an HTML file in the same directory as the output binary. The callgraph gives definition and reference information for all functions in the image.

   See the description of the `--callgraph` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information and a description of the output.

6. Select the link information you want to view. These options instruct the linker to generate size information for sections in the output image:

   Sizes Gives a list of the Code and Data (RO Data, RW Data, ZI Data, and Debug Data) sizes for each input object and library member in the image.

   Totals Gives totals of the Code and Data (RO Data, RW Data, ZI Data, and Debug Data) sizes for input objects and libraries.

   Unused Lists all unused sections that were eliminated from the image.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Veneers Gives details of the linker-generated veneers. See the *RealView Compilation Tools Linker and Utilities Guide* for more information on linker-generated veneers, such as interworking veneers.

See the description of the `--info` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

7. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring linker diagnostics

Use the Diagnostics panel to configure linker diagnostic options:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Linker** in the Target Settings Panels list and click the **Diagnostics** tab to display the configuration panel (Figure 10-31).

Figure 10-31 RealView Linker Diagnostics configuration panel

3. Enter a tag, or a list of tags separated by commas, as required:

Suppress Disables all diagnostic messages that have the specified tag(s).

Set to warning

Sets the diagnostic messages that have the specified tag(s) to Warning severity.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Set to error

Sets the diagnostic messages that have the specified tag(s) to Error severity.

4. Click **Apply** to apply your settings without closing the Target Settings window.

Configuring linker extras

Use the Extras panel to configure extra linker options:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Linker** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 10-32).

Figure 10-32 RealView Linker Extras configuration panel

3. Select options, as required:

Make undefined symbols refer to

Enter a symbol to match each reference to an undefined symbol in your code. The symbol must be both defined and global, otherwise the link step fails.

This option is useful during top-down development. It enables you to test a partially-implemented system by matching references to missing functions to a dummy function.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

See the description of the `--unresolved` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information.

Via file name

Enter a filename, or click **Choose…** to select a via file from the standard file dialog box.

You can use a via file to specify additional linker options. See the *RealView Compilation Tools Linker and Utilities Guide* for more information on using via files with the linker.

4. Click **Apply** to apply your settings without closing the Target Settings window.

10.6.2 Configuring RealView fromelf

The RealView `fromelf` utility can perform a number of format conversions on linker, compiler, or assembler output, such as:

- converting an ELF image file to a binary format suitable for embedding in ROM
- disassembling output, and extracting other information such as object sizes, symbol and string tables, and relocation information.

To use and configure `fromelf` you must specify it in the Post-linker field of the Target Settings configuration panel. See *Configuring target settings* on page 10-8 for information. When you select this option, a RealView FromELF configuration panel is added to the Linker group of panels.

This section describes how to configure the RealView FromELF (`fromelf`) from within the CodeWarrior IDE. It provides general descriptions of the available fromelf options.

The RealView FromELF panel consists of a number of tabbed configuration panels. The available options are described in:

- *Configuring output format* on page 10-77
- *Configuring text format* on page 10-78
- *Configuring fromelf diagnostics* on page 10-79
- *Reading fromelf options from a file* on page 10-80

These sections provide general descriptions of the available assembler options. For more detailed descriptions, see the *RealView Compilation Tools Linker and Utilities Guide*.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Configuring output format

To configure the output produced from the `fromelf` utility:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView FromELF** in the Target Settings Panels list and click the **Output** tab to display the configuration panel (Figure 10-33).

Figure 10-33 RealView FromELF Output configuration panel

3. Click the **Output format** drop-down menu and select the required format. Either:
   - Select **Text** to extract a text file of information about the output image. With this option selected, a **Text** tab is displayed to enable you to select the text format (see *Configuring text format* on page 10-78).
   - Select from the binary output options, for example, choose **ELF** to resave a with-debug ELF image as a no-debug ELF image. See the *RealView Compilation Tools Linker and Utilities Guide* for details of the available options.

4. If you select **ELF** as your output format, the **Include debug sections in output** option is enabled. Select this to ensure that debug table information is included in the `fromelf` output.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

5. Enter the path name of the output file or click **Choose…** to select an output file from the standard file dialog box. Be aware that:
   - If your project is based on the ARM stationery and you select **TEXT** as your output format, the output file is automatically named `default`. The contents are formatted for easy reading.
   - If you select **TEXT** as your output format and you do not enter a path name, output text information is displayed in an Errors and Warnings message window. However, all formatting is removed.
   - Binary output cannot be displayed in an Errors and Warnings message window. If you select from the binary output options, for example, **ELF**, you must specify the path name of the output file (there is no default in the project stationery).
6. Click **Apply** to apply your changes. When you make your project, the CodeWarrior IDE calls `fromelf` to process the output.

Configuring text format

If you select **Text** from the **Output format** drop-down menu on the Output configuration panel (see *Configuring output format* on page 10-77), a **Text** tab is displayed to enable you to configure the format for text file output:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView FromELF** in the Target Settings Panels list and click the **Text** tab to display the configuration panel (Figure 10-34 on page 10-79).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-34 RealView FromELF Text configuration panel

3. Select one or more text format flags to choose the information contained in the text file.

   If a specific text category is not specified, the default is to output header information.

   For more information about these options see the `--text` command-line option in the *RealView Compilation Tools Linker and Utilities Guide*.

4. Click **Apply** to apply your changes without closing the Target Settings window.

Configuring fromelf diagnostics

To control the output of the diagnostic messages generated by `fromelf`:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView FromELF** in the Target Settings Panels list and click the **Diagnostics** tab to display the configuration panel (Figure 10-35 on page 10-80).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-35 RealView FromELF Diagnostics configuration panel

3. Enter a tag, or a list of tags separated by commas, as required:

   Suppress Disables all diagnostic messages that have the specified tag(s).

   Set to warning

   Sets the diagnostic messages that have the specified tag(s) to Warning severity.

   Set to error

   Sets the diagnostic messages that have the specified tag(s) to Error severity.

4. Click **Apply** to apply your changes without closing the Target Settings window.

Reading fromelf options from a file

Use the Extras configuration panel to specify a via file for the `fromelf` utility. A via file is a text file that contains additional command-line arguments to fromelf. Via files are read when fromelf is invoked. The via file options are processed after the options specified in the configuration panels, and override them if there is a conflict. You can use via files:

- to ensure that the same settings are used by different build targets and projects
- to include exceptionally long command-line options to fromelf.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

See the *RealView Compilation Tools Compiler User Guide* for a description of via file syntax.

To specify a via file:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView FromELF** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (see Figure 10-36).

Figure 10-36 RealView FromELF Extras configuration panel

3. Enter the path name of the via file you want to use, or click **Choose…** and select the via file from the standard file dialog box.
4. Click **Apply** to apply your changes without closing the Target Settings window.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.7 Configuring editor settings

This section describes changes you can make to the CodeWarrior IDE editor that apply to the current build target. See *Choosing editor preferences* on page 9-21 for information on setting global editor preferences.

10.7.1 Custom Keywords

The Custom Keywords configuration panel enables you to set colors for defined sets of keywords. The colors you define are used to display the keywords in the CodeWarrior IDE editor. Custom keywords are project-specific, not global to the CodeWarrior IDE. See *Text Colors* on page 9-32 for more information on:

- setting global keyword sets
- setting color options
- importing and exporting keyword sets.

To configure custom keyword sets:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **Custom Keywords** in the Target Settings Panels list to display the configuration panel (Figure 10-37)

Figure 10-37 Custom Keywords configuration panel

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

3. Change the keyword sets, as required:
   - To change the color for the keyword set, click the color sample and select the color you want from the standard Windows color picker.
   - To change the contents of a keyword set, click **Edit…** and make the appropriate entries in the dialog box.

     To delete a keyword from the list, select it and press the Backspace key.

     You can also export and import sets of keyword definitions from this dialog. Keyword files are alphabetically sorted text files with one line for each keyword. Each line is terminated with a carriage return.

     If you click **Factory Settings**, the keyword colors are restored to the defaults but keyword lists are not affected, that is, their contents are retained.
4. Click **Apply** to apply your changes without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.8 Configuring the debugger

The Debugger group of panels enables you to configure debugger settings in the CodeWarrior IDE (see Figure 10-38 on page 10-85). There are a number of areas of functionality that are not implemented by the ARM version of the CodeWarrior IDE. This means that selected panels in this group are *not* supported in CodeWarrior for RVDS, that is:

- **Analyzer Connections**
- **Other Executables**
- **Debugger Settings**
- **Remote Debugging**.

This section describes how to configure options on panels in this group that are supported in CodeWarrior for RVDS:

- **RealView Debugger**, to configure the debugger, see *Configuring the RealView Debugger*.
- **RealView Runner**, to configure the RealView Debugger that is called to execute, rather than debug, an image file (see *Configuring the RealView Runner* on page 10-85).

Note

The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately.

10.8.1 Configuring the RealView Debugger

You can configure the CodeWarrior IDE to call the RealView Debugger to load and debug your images.

This section describes how to configure the RealView Debugger to debug and run executable images built from CodeWarrior IDE projects. See Chapter 5 *Working with the RealView Debugger* for more information.

See the *RealView Debugger User Guide* for more information.

Specify arguments for the RealView Debugger using the **Arguments** tab:

1. Display the Target Settings Panels for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).
2. Click **RealView Debugger** in the Target Settings Panels list and check the to display the configuration panel (Figure 10-38 on page 10-85).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure 10-38 RealView Debugger Arguments configuration panel

3. Enter any command-line arguments required by your executable image in the **Arguments for image file** text box.

4. Click **Apply** to apply your settings without closing the Target Settings window.

10.8.2 Configuring the RealView Runner

Use the RealView Runner panel to choose the debugger that the CodeWarrior IDE calls when you select **Run** from the **Project** menu, that is, the debugger that is called to *execute*, rather than *debug*, an image file. You can use the RealView Debugger to run executable images.

You can specify different debuggers to be called when you debug, and when you run.

The RealView Runner panel consists of a number of tabbed configuration panels. The tabs and options for this panel are exactly the same as for the RealView Debugger panel. See *Configuring the RealView Debugger* on page 10-84 for information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

10.9 Configuring miscellaneous features

The **Miscellaneous** entry in the Target Settings Panels list describes a single panel to configure miscellaneous options, see *RealView Disassembly Features*.

For more information about disassembling code, see *Disassembling code* on page 4-44.

10.9.1 RealView Disassembly Features

This panel enables you to configure the settings that are passed to `fromelf` when you right-click on an object file in the project window and select **Disassemble**. To configure the disassembly settings:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 10-4).

2. Click **RealView Disassembly** in the Target Settings Panels list to display the configuration panel (Figure 10-39).

Figure 10-39 RealView Disassembly configuration panel

3. Select the options you require from the Text format flags group box. The Disassemble code option is selected by default.

   The options for this panel are exactly the same as for the **Text** tab in the RealView FromELF configuration panel. See *Configuring text format* on page 10-78.

   For more information about these options see the `--text` command-line option in the *RealView Compilation Tools Linker and Utilities Guide*.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

4. Click the **Diagnostics** tab to configure the diagnostic information produced.

   The options for this panel are exactly the same as for the **Diagnostics** tab in the RealView FromELF configuration panel. See *Configuring fromelf diagnostics* on page 10-79.

5. Click the **Extras** tab to configure additional options.

   The options for this panel are exactly the same as for the **Extras** tab in the RealView FromELF configuration panel. See *Reading fromelf options from a file* on page 10-80.

6. Click **Apply** to apply your changes without closing the Target Settings window.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Appendix A
Running the CodeWarrior IDE from the Command Line and Command Window

This appendix explains how to use the CodeWarrior IDE from the command line and Command Window. It contains the following sections:

- *Using the CodeWarrior IDE from the command line* on page A-2
- *Using the CodeWarrior Command Window* on page A-4.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

A.1 Using the CodeWarrior IDE from the command line

CodeWarrior can be run from the command line using the command `CmdIDE.EXE`. `CmdIDE.EXE` is a console window program that can be started from a DOS prompt to build project files that have been created and edited with the CodeWarrior IDE.

`CmdIDE.EXE` is located in:

install_directory`\IDEs\CodeWarrior\CodeWarrior\...\...\win_32-pentium\bin`

Change directory to this location to run `CmdIDE.EXE`. This location is not added to your **PATH** when you install RVDS. `CmdIDE.EXE` invokes the CodeWarrior IDE, passes the correct parameters to produce a build, and waits for the CodeWarrior IDE to finish its operation.

The command-line arguments are:

ProjectName Specifies the project to use.

`/t` *TargetName*
: Specifies a target to become the current target.

`/r` Removes the objects of the current target before building.

`/b` Builds the current target.

`/c` Closes the project after building.

`/q` Quits the CodeWarrior IDE after building.

`/v[y|n|a]` Options for converting projects on open:
- `y` Convert without asking.
- `n` Do not convert.
- `a` Ask whether to convert.

`/s` Forces the command line to be processed in a new instance of the CodeWarrior IDE (rather than using a current instance).

If more than one project document is specified to be opened in the command line, the `/t` target and `/b` build command flags apply to the first project document found in the list of documents.

If no project is specified in the command line, it uses the usual logic in the CodeWarrior IDE (front project or default project) to select which project to build.

For example, to start the CodeWarrior IDE, open and then build the example Dhrystone project, enter:

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

```
cmdide "install_directory\RVDS\Examples\dhrystone\dhrystone.mcp" /t Debug /c /b
```

If your installation directory has spaces, then you must enclose it in double quotes.

If no target is specified, `cmdIDE` uses whatever the current target is for that project.

All build commands are executed using a different process from the one launched from the command line. The original process returns when the command line has been completely processed and the build has completed.

The following codes are returned and can be tested using the `IF ERRORLEVEL` instruction in a batch file:

`0`	No error
`1`	Error opening file
`2`	Project is not open
`3`	IDE is already building
`4`	Invalid target name (for `/t` flag)
`5`	Error changing current target
`6`	Error removing objects
`7`	Build was canceled
`8`	Build failed
`9`	Process aborted.

Note

You can use the command `IDE.EXE` to run the CodeWarrior IDE. `IDE.EXE` understands the same parameters as `CmdIDE.EXE`. However, it is particularly important on Windows 95, Windows 98, and Windows ME to use `CmdIDE.EXE` to ensure that builds are correctly serialized rather than executed all at once.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

A.2 Using the CodeWarrior Command Window

A Command Window can be opened from the CodeWarrior IDE. You can issue commands from this window to control many of the features of CodeWarrior.

A.2.1 Opening the Command Window

To open the Command Window, select **Command Window** from the **View** menu. The Command Window is displayed (Figure A-1).

Figure A-1 Command Window

Commands can be entered at the command prompt, %>.

A list of available commands is shown at the bottom of the Command Window. Pressing the spacebar scrolls through all the commands which are available. The highlighted areas of each command is the shortcut that can be used for that command.

For example, to show information about the current project, enter the `project` (or `proj`) command (see Figure A-2 on page A-5).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure A-2 Running the project (proj) command

To display information about all the commands available including a description of each command, enter the command `help` or `h`.

To display detailed information about a particular command including the syntax of the command, enter the command `help` followed by the command name. For example:

```
help log
```

A.2.2 Supported commands

Table A-1 describes all the commands which are available from the CodeWarrior IDE Command Window.

If a command is not supported in the ARM® version of the CodeWarrior IDE for RVDS, this is indicated by *Not used*.

Table A-1 CodeWarrior commands from the Command Window

Command	Description
alias	Create, remove or list alias.
bp	*Not used*
bringtofront	Console window always on top.
cd	Change directory.
change	*Not used*

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table A-1 CodeWarrior commands from the Command Window (continued)

Command	Description
cls	Clear screen.
config	Configure the Command Window.
copy	*Not used*
debug	Debug a project. This command will `Make` the project first, if required, to create an image and then initialize the currently selected debugger for the project.
dir	Display the contents of the current directory. (This command is the same as the DOS `dir` command.)
disassemble	*Not used*
display	*Not used*
evaluate	*Not used*
exit	Close the Command Window.
getpid	*Not used*
help	Display help information.
history	List command history.
kill	Close the current debug session.
log	Log commands and/or session to a log file.
`Make`	Build the specified project or the default project if no project name is specified.
next	*Not used*
project	Open or close a project file.
pwd	Display current working directory.
quitIDE	Quit the CodeWarrior IDE.
radix	*Not used*
removeobj	Remove all binaries for the specified project or the default project if no project is specified.
reset	*Not used*

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table A-1 CodeWarrior commands from the Command Window (continued)

Command	Description
restart	Restart a debugging session.
restore	*Not used*
rget	*Not used*
rset	*Not used*
save	*Not used*
sourcedisplay	*Not used*
stack	*Not used*
status	*Not used*
step	*Not used*
stop	*Not used*
switchtarget	*Not used*
system	Execute a system command.
view	*Not used*
wait	*Not used*
watchpoint	*Not used*
window	*Not used*

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Appendix B
CodeWarrior IDE Installation and Preference Settings

This appendix describes how to install multiple copies of the CodeWarrior IDE, and how to use the CodeWarrior IDE for RVDS with other versions of the CodeWarrior IDE. It contains the following sections:

- *The CodeWarrior IDE preferences directory* on page B-2
- *Using different versions of the CodeWarrior IDE* on page B-3
- *Using the cwfileassoc tool* on page B-4.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

B.1 The CodeWarrior IDE preferences directory

The CodeWarrior IDE maintains preferences and persistence information in the following location:

```
c:\Documents and Settings\<username>\Local Settings\Application Data\Metrowerks
```

The preferences file is created when the CodeWarrior IDE is started for the first time, if it does not already exist.

Note

If you want to install multiple copies of the CodeWarrior IDE for RVDS you can preconfigure a single installation and copy the preferences directory for that installation to each machine.

CodeWarrior IDE preferences are not deleted when you uninstall the CodeWarrior IDE for RVDS.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

B.2 Using different versions of the CodeWarrior IDE

The CodeWarrior IDE for RVDS is customized to support the ARM® toolchain. This means that:

- your CodeWarrior IDE preferences might not be applicable to other CodeWarrior IDE versions
- some components of the CodeWarrior IDE that are registered in the Windows registry at installation are specific to the CodeWarrior IDE for RVDS.

To switch from the CodeWarrior IDE for RVDS to another version of the CodeWarrior IDE on the same machine:

1. Rename the CodeWarrior IDE Preferences file.
2. Run the `regservers.bat` batch file for the installation you want to use. Typically `regservers.bat` is located in:

   install_directory`\IDEs\CodeWarrior\CodeWarrior\...\...\win_32-pentium\bin`

To switch back to the CodeWarrior IDE for RVDS, rename your preferences directories and run `regservers.bat` again.

Note

The CodeWarrior IDE for RVDS v3.1 uses the `RVCT31BIN` environment variable to locate the RVCT build tools, not the `PATH` environment variable. This is set up for you by the RVDS installer.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

B.3 Using the cwfileassoc tool

The `cwfileassoc` tool can be used to modify the CodeWarrior file associations. If you chose to install CodeWarrior during the installation of RVDS, you were prompted to specify which file types to associate with CodeWarrior. After installation, you can change the file associations using the `cwfileassoc` tool.

B.3.1 cwfileassoc command syntax

You can find the `cwfileassoc` tool at the following location:

install_directory\IDEs\CodeWarrior\CodeWarrior*version_number**build_number*

The command syntax of the `cwfileassoc` tool is:

`cwfileassoc "`*install_directory*`"`

If your installation directory has spaces, then you must enclose it in double quotes.

Before using this command, you should ensure that the CodeWarrior IDE is not running on your computer.

B.3.2 cwfileassoc example

To modify the CodeWarrior file associations with RVDS installed in the default location, enter `cwfileassoc "C:\Program Files\ARM"`.

You are prompted to modify each of the following file types in turn:

- `.mcp`
- `.c`
- `.h`
- `.cpp`
- `.hpp`
- `.s`
- `.scat`.

At each file type prompt, enter one of the following letters:

S To associate the file type with CodeWarrior.

R To remove the association of the file type from CodeWarrior.

A To leave the association for the file type unchanged.

The new file associations will come into effect when the `cwfileassoc` tool has exited.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Appendix C
CodeWarrior Project Conversion Utilities

The CodeWarrior IDE for RVDS v3.1 provides utilities which you can use for converting projects or files created using other development environments into CodeWarrior for RVDS projects, or for exporting CodeWarrior RVDS projects into other file formats.

This appendix describes how to convert:

- CodeWarrior for ARM Developer Suite™ (ADS) projects. See *Converting old CodeWarrior projects* on page C-2.
- GNU `Make` files. The CodeWarrior IDE can export CodeWarrior IDE projects as GNU `Make` files. See *Converting GNU Make files* on page C-5.
- Projects to use external `Make` files. See *Using an external Make file* on page C-6.
- RealView® Debugger project files. See *Converting RealView Debugger project files* on page C-14.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.1 Converting old CodeWarrior projects

If you have created projects using a previous version of CodeWarrior, you can upgrade them to the latest CodeWarrior for RVDS projects. This section describes how to do this.

To convert your old CodeWarrior projects to the latest CodeWarrior for RVDS projects:

1. Select **Open...** from the **File** menu to display the Open dialog box.

2. Locate your old CodeWarrior project file (`.mcp`), and click **Open**.

   You can select more than one file if required. For example, if you want to convert both the interwork project files `ARM_to_Thumb.mcp` and `Thumb_to_ARM.mcp`, press the Ctrl key then select each project file.

   A Convert Project dialog box is displayed, that asks if you want to update the Target Settings panels to the latest CodeWarrior for RVDS settings.

Figure C-1 Convert Project dialog box

3. If you have selected a project file to convert that contains sub-projects, select **Use For All Remaining Projects** to update the Target Settings panels for all the old CodeWarrior project files. This prevents the Convert Project dialog box being displayed again when the CodeWarrior IDE converts the sub-project files.

4. Click **OK** to convert the project. The CodeWarrior IDE:
   - converts the project
   - renames the original project file to *`filename`*`.old.mcp`
   - if necessary, creates a subdirectory called *`ProjectName`*`_Data`, which also contains subdirectories for each build target defined in the original project
   - creates a text file with the filename *`ProjectName TargetName`* `compiler command lines.txt`, for each build target containing the build options used.

   Note

   The ARM® compiler, `armcc` , now enables you to compile your ARM and Thumb, C and C++ code. The latest CodeWarrior IDE for RVDS contains only one compiler panel.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Only the settings from the first non-empty command line for the ARM C, ARM C++, Thumb C, and Thumb C++ compilers are used for the converted CodeWarrior for RVDS project. Where multiple distinct compilers are found, the settings for all non-empty old compiler panels are stored in the `ProjectName TargetName compiler command lines.txt` files. You can view the contents of these files to determine any missing settings that you might want to re-apply to your project.

If you click **Cancel** for a project, then that project is not converted. If you selected multiple projects to convert, then the dialog box is displayed again for the next project.

Note

Old CodeWarrior projects contain the `Release`, `DebugRel`, and `Debug` build targets. These build targets are included in your converted project. However, the ARM project stationery provided with CodeWarrior for RVDS provides only the `Release` and `Debug` build targets (see *Predefined build targets* on page 4-4).

Some build tool options cannot be imported from old CodeWarrior projects. See *Fixing options that cannot be imported from old CodeWarrior projects* for more information.

C.1.1 Fixing options that cannot be imported from old CodeWarrior projects

If your old CodeWarrior project uses options that cannot be imported, they are highlighted in red in the equivalent command line for the associated panel. To fix these, do the following for each of the build targets `Release` and `Debug`:

1. Select ***TargetName* Settings…**from the **Edit** menu.
2. Click `RealView Assembler` in the Target Settings Panels list. The CodeWarrior IDE displays the RealView Assembler panel.
3. Remove any options from the Equivalent Command Line that are highlighted in red.
4. Click `RealView Compiler` in the Target Settings Panels list. The CodeWarrior IDE displays the RealView Compiler panel.
5. Remove any options from the Equivalent Command Line that are highlighted in red.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

6. Review the remaining compiler options in the equivalent command line and verify that they are correct. In particular, if your old CodeWarrior project is an ARM/Thumb interworking project, only the settings for the ARM C compiler are imported. For a Thumb project only:

   a. Change the Initial State to **Thumb**.

      This adds the `--thumb` compiler option.

7. Click `RealView FromELF` in the Target Settings Panels list. The CodeWarrior IDE displays the RealView FromELF panel.

8. Remove any options from the Equivalent Command Line that are highlighted in red.

9. Review the remaining `fromelf` options in the equivalent command line and verify that they are correct.

10. Click **OK** to save your changes and close the Target Settings window.

    When you have made the appropriate changes to all the build targets, you can build your project.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.2 Converting GNU `Make` files

This section describes how to export CodeWarrior IDE project files to GNU `Make` files. To do this:

1. Open the CodeWarrior IDE project you want to convert.
2. Select **Export Project as GNU Make file** from the **Project** menu.

   The project is converted, and the make file is saved in the same location as the CodeWarrior IDE project file.

The exported `Make` file is supported by the GNU BASH shell v2.05 and GNU Make v3.80 utilities from the following distributions:

- Cygwin (`http://www.cygwin.com/`)
- MinGW (`http://www.mingw.org/`).

C.2.1 Importing GNU `Make` files

If you are upgrading to RVDS v3.1 from a pre-RVDS v3.0 release, be aware that GNU `Make` files cannot be imported into CodeWarrior for this release. The CodeWarrior IDE for RVDS v2.2 contained a `Make` file Importer wizard. However, this feature is not supported.

You can now use the External Build Wizard to create a CodeWarrior project that uses an external GNU `Make` file (see *Using an external Make file* on page C-6 for more information).

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.3 Using an external `Make` file

The External Build Wizard enables you to create a CodeWarrior IDE project that makes use of an external command line driven build system, such as GNU `Make`. The wizard associates the build target in the newly-created project with a command line that is executed when the build command is invoked from the CodeWarrior IDE.

You can optionally add files used by your external build system to the CodeWarrior IDE project in the usual way. This enables you to organize source code files and to make use of the project management features, such as code completion and text editing, while still building with your original `Make` file.

Note

Make sure that the location of your `Make` file executable is included in your `PATH` environment variable.

Any executable file that you create can be debugged with the RealView Debugger in the usual way (see Chapter 5 *Working with the RealView Debugger*).

The procedure described in this section assumes that your `Make` file has the following build targets:

- `image1.axf`, to build `image1.axf`
- `image2.axf`, to build `image2.axf`
- `all`, to build all images
- `clean`, to remove all object and image files
- `rebuild`, to remove the object and image files, and rebuild all images.

Note

You can add only a single build target with the External Build Wizard. If you want to include additional build targets, you must add these manually after the project is created. This example shows how to add the `image1.axf` build target.

The procedure includes the steps described in the following sections:

1. *Creating an external build project* on page C-7.
2. *Configuring an external debugger* on page C-11.
3. *Adding extra build targets to a project* on page C-12.
4. *Building a target* on page C-13.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.3.1 Creating an external build project

To create a new CodeWarrior project that builds using an external build file:

1. Select **New…** from the **File** menu. The CodeWarrior IDE displays a New dialog box (see Figure C-2).

Figure C-2 New external project

2. Ensure that the **Project** tab is selected and select `External Build Wizard`.
3. Click **Set…** to display the Create New Project dialog box.

   By default, the Save as type field is set to `Project Files (.mcp)`. Leave this setting as it is.
4. Locate the directory for your project (`C:\Myprojects\my_app` in this example).
5. Enter a file name for your project.

   For this example, enter **my_app**.
6. Ensure that the **Create Folder** checkbox is unchecked, so that the existing directory structure is used for the project.
7. Click **Save** to close the Create New Project dialog box.

   The New dialog box is updated as follows:
   - the Project name is set to `my_app`

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

- the Location is set to the full path for the project, `C:\Myprojects\my_app` in this example.

8. Click **OK** to continue. An information page is displayed.

9. Click **Next >** to configure the build target options (Figure C-3).

Figure C-3 Setup external project

10. Specify the Build Command to use.

Note

Make sure you use the same case as the target name in your `Make` file.

For this example, specify the following Build Command:

```
Make -f build.mk image1.axf
```

Note

You do not have to specify an Output file name.

11. Click **Finish** to see a summary of the settings (Figure C-4 on page C-9).

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure C-4 Generate external project

12. Click **Generate** to create the CodeWarrior project.

    For this example, CodeWarrior uses the project name you specified to create the following:

    - A project file in the project directory, `my_app.mcp`
    - A data directory in the project directory, `my_app_Data`.
    - A `my_app` build target.

13. Optionally, rename the build target to indicate the image that it builds:

    a. Click the **Target** tab to display the build target list.

    b. Double-click on the build target (`my_app` in this example). The project Settings window is displayed. Figure C-5 on page C-10 shows the Settings window for this example.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Figure C-5 External target settings

c. Ensure that the `Target Settings` panel is selected in the left pane.

d. Change the Target Name as required. For this example, change the name to `image1.axf`.

e. Click **OK** to save the changes.

You can use this build target name if you do not want to have any other build targets. Although no additional configuration is required, you can:

- Add your source files to the project (see *Adding files to a project* on page 3-26). This enables you to open and edit the sources in CodeWarrior. However, the CodeWarrior internal build features associated with referenced source files are not available.
- Configure an external debugger, such as RealView Debugger, to debug your application image. This enables you to launch the debugger from within CodeWarrior (see *Configuring an external debugger* on page C-11).
- Create a build target for each target in your `Make` file. For instructions on how to do this, see *Adding extra build targets to a project* on page C-12.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.3.2 Configuring an external debugger

To configure an external debugger, such as RealView Debugger:

1. Click the **Target** tab for the project to display the build target list.
2. Double-click on the build target (`image1.axf` in this example). The project Settings window is displayed. Figure C-5 on page C-10 shows the Settings window for this example.
3. Select the Build Extras panel in the left pane to display the Build Extras settings, shown in Figure C-6.

Figure C-6 Build extras settings

4. Select **Use External Debugger**. The related fields are enabled.
5. Set the Application field to the full path of the debugger executable. Click the Application **Browse...** button to locate the debugger executable.

   For example, to use RealView Debugger, the executable is located in:

   `install_directory\RVD\Core\...\bin\rvdebug.exe`

   where ... indicates the version and build number for your RealView Debugger installation, which you must specify.
6. Specify any debugger command line arguments, if required.

   See the *RealView Debugger Command Line Reference Guide* and the *RealView Debugger User Guide* for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.3.3 Adding extra build targets to a project

The easiest method to include additional build targets is to copy the existing build target, and then modify the settings. The following procedure assumes that you already have the `image1.axf` build target created and configured in the previous sections and you now want to add the `image2.axf` build target.

To include an additional build target in your external build project:

1. Click the **Target** tab for the project to display the build target list.
2. Select an existing build target (`image1.axf` in this example).
3. Select **Create Target...** from the **Project** menu. The New Target dialog box is displayed, shown in Figure C-7.

Figure C-7 New Target dialog box

4. Create the new target as follows:
   a. Enter a new name for the target. For this example, enter **image2.axf**.
   b. Select **Clone Existing Target**.
   c. Select the target to be cloned (`image1.axf` in this example).
   d. Click **OK**. The new target is created.
5. Configure the new target as follows:
   a. Double-click on the new build target (`image2.axf` in this example). The project Settings window is displayed.
   b. Select **External Build** in the Target Settings Panels to display the External Build settings.
   c. Change the Build Command to:
      ```
      Make -f build.mk image2.axf
      ```
   d. If you have set up an external debugger as described in *Configuring an external debugger* on page C-11, then change the settings as required.
   e. Click **OK** to save your changes.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

You can also use this method to create build targets for the `clean`, `rebuild`, and `all` targets in the `Make` file for this example. However:

- for the `clean`, `rebuild`, and `all` targets, uncheck the **Use External Debugger** checkbox on the Build Extras panel (see Figure C-6 on page C-11 for example)
- use the `all` target to build both the `image1.axf` and `image2.axf` images, but use the build target for the corresponding image to start the debugger.

C.3.4 Building a target

To build the required target:

1. Right-click on the required build target, for example `image1.axf`, to display the context menu.
2. Select **Make** from the context menu.

Note

You cannot use the **Pre-linker** and **Post-linker** options on the Target Settings panel in an external build project (see *Configuring target settings* on page 10-8). You must include the appropriate pre-linker and post-linker in your project's `Make` file.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

C.4 Converting RealView Debugger project files

All types of RealView Debugger projects can be converted to CodeWarrior projects. The conversion procedure to follow depends on the type of RealView Debugger project to be converted, see:

- *Converting RealView Debugger Standard and Library projects*
- *Converting RealView Debugger Container projects* on page C-18.

Note

You can use an RealView Debugger Custom project to specify your own `Make` file, or to specify an external build program or script, or to use no build model. You cannot convert an RealView Debugger Custom project to a CodeWarrior project, or import the project `Make` file directly. See *Using an external Make file* on page C-6 for information on how to access GNU `Make` files using the External Build wizard.

C.4.1 Converting RealView Debugger Standard and Library projects

To convert RealView Debugger Standard and Library project files you must convert the project to CodeWarrior RVDS XML format using the `prj2xml` conversion utility. You can then import project files that are coded in CodeWarrior for RVDS XML format into the CodeWarrior IDE. This section includes:

- *Converting a RealView Debugger project file to XML format*
- *prj2xml command options* on page C-16
- *Example conversion of dhrystone.prj* on page C-17
- *Importing XML formatted project files* on page C-18.

For more information on importing and exporting XML files in CodeWarrior, see *Importing and exporting a project as XML* on page 3-17.

Converting a RealView Debugger project file to XML format

To convert an RealView Debugger project file to XML, run the `prj2xml` command and specify the RealView Debugger project file (`.prj`) to be converted. You can use the `prj2xml` command to convert one or more RealView Debugger project files to CodeWarrior for RVDS XML formatted files.

The syntax of the `prj2xml` command is:

```
prj2xml [options] {--output=filename project} | {project1 project2 ... projectN}
```

If you want to convert more than one project file, you must not use the `--output` option. In this case, the converted project files are named *`projectfile`*`.prj.xml`. The converted files are each placed in the same location as the original project files.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

See Table C-1 on page C-16 for details of the options supported by the `prj2xml` command.

By default, the `prj2xml` conversion utility:

- accepts relative and absolute path names for the output directory
- accepts target-specific output filenames
- takes into consideration the RealView Debugger subdirectory rule (in the `CONFIGURATION` group).

However, if the RealView Debugger project includes relative path names and the CodeWarrior IDE project is created in a different location, these paths are broken. If you want to create the CodeWarrior IDE project in a separate directory from the original RealView Debugger project then you must either:

- replace any relative paths in the RealView Debugger project with absolute paths before converting the project to XML
- replicate the directory structure in the new location.

The CodeWarrior IDE build target names are created using the original build target names in each RealView Debugger project file, and are prefixed with the `COMPILE=`, `ASSEMBLE=`, and `CUSTOM=` group names. For example, if your RealView Debugger project has `Debug` and `Release` build targets, and ARM C++ sources (in the `COMPILE=arm_cpp` group), then the CodeWarrior IDE build target names are `arm_cpp_Debug` and `arm_cpp_Release`.

If the RealView Debugger project includes debug controls and optimization levels for the compiler, the `prj2xml` conversion utility:

- makes no change to any multi-optimization setting explicitly set for any build target
- defaults to `-O0` for the `Debug` target if `-g` is set but no optimization level is explicitly set.

  If a target includes other multi-optimization settings, for example `-Ospace` or `-Otime`, then defaulting to `-O0` has no effect (see `target = "arm_Debug"` in *Example conversion of dhrystone.prj* on page C-17 where `-O0` has been added during conversion).

  For information on how these options interact, see the *RealView Compilation Tools Compiler User Guide*.

See *Configuring debug and optimization* on page 10-59 for more information on configuring these settings for the compiler.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

prj2xml command options

Table C-1 shows the options supported by the `prj2xml` command.

Table C-1 prj2xml command options

Option	Description
`-h --help`	Displays help on the `prj2xml` command options.
`-v --version`	Displays the version of the `prj2xml` command.
`-q --quiet`	Suppresses the status information that is normally displayed during the conversion. See *Example conversion of dhrystone.prj* on page C-17 for an example of the status information that is displayed.
`-o`*`file`* `--output=`*`filename`*	Enables you to give the output XML file a specific filename. **Note** You can specify only one RealView Debugger project file if you use this option.
`--template=`*`filename`*	The XML template file specification. The `template.xml` file supplied with `prj2xml` is used by default.
`--spec=`*`filename`*	The RealView Debugger project file specification you want to use. The `spec.stp` file supplied with `prj2xml` is used by default.

Note

You can use the options `--template` and `--spec` (see Table C-1) to override the default configuration files. These options are not normally required. If you use these options, you must use them with care.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Example conversion of dhrystone.prj

This example shows how to convert the `dhrystone.prj` file, with the status information displayed:

```
> prj2xml dhrystone.prj
    read from "dhrystone.prj"
    output to "dhrystone.prj.xml"

        [arm]
            target = "arm_Release"
                LINKER command line = " "
                TARGET command line = " --outputname=”dhrystone" --no_useprojectname
                ASSEMBLE command line = ""
                COMPILE command line = " -g -O3 -DMSC_CLOCK -W --no_inline -Otime"
                file = "dhry_2.c"
                file = "dhry_1.c"
                path = ".\"

            target = "arm_Debug"
                LINKER command line = " "
                TARGET command line = " --outputname=”dhrystone" --no_useprojectname
                ASSEMBLE command line = ""
                COMPILE command line = " -g -DMSC_CLOCK -W -O0 -Otime"
                file = ".\dhry_2.c"
                file = ".\dhry_1.c"
                path = ".\"

            target = "arm_DebugRel"
                LINKER command line = " "
                TARGET command line = " --outputname=”dhrystone” --no_useprojectname"
                ASSEMBLE command line = ""
                COMPILE command line = " -O1 -g -DMSC_CLOCK -W -Otime"
                file = ".\dhry_2.c"
                file = ".\dhry_1.c"
                path = ".\"
```

The names of the `COMPILE`, `ASSEMBLE`, and `CUSTOM` groups are used as the prefix for the build target names in the converted file. In this example, the `[COMPILE=arm]` group has been used for the `arm_Release`, `arm_Debug`, and `arm_DebugRel` build targets.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Importing XML formatted project files

To import an XML formatted project file:

1. Select **Import Project...** from the **File** menu. An Open dialog box is displayed.
2. Locate the required XML formatted project file.
3. Click **Open**. A Name new project as dialog box is displayed.
4. Enter a filename for the new project.
5. Click **Save**.

   The XML project file is imported, and converted to a CodeWarrior for RVDS project file.

   The CodeWarrior IDE opens the project and displays it in a project window

   The project contains the build targets, source files, and settings specified in the XML file.

C.4.2 Converting RealView Debugger Container projects

The `prj2xml` command cannot be used to directly convert RealView Debugger Container projects. (RealView Debugger Container are projects which contain other RealView Debugger sub-projects.) If you want to convert an RealView Debugger Container project then you must convert each sub-project in the Container project as a separate project. The procedure to follow is:

1. Locate each sub-project (`.prj` file) of the RealView Debugger Container project.
2. Convert and import each `.prj` file of the RealView Debugger Container project into the CodeWarrior IDE using the `prj2xml` utility, as described in:
   a. *Converting a RealView Debugger project file to XML format* on page C-14.
   b. *Importing XML formatted project files*.
3. Create a new CodeWarrior IDE project and add all the converted sub-projects to it.

   For more information on creating sub-projects in the CodeWarrior IDE, see *Working with multiple build targets and sub-projects* on page 3-40.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Appendix D
CodeWarrior IDE Reference

This appendix describes each menu command in the CodeWarrior IDE and the default key bindings for those commands. You can use this appendix as a convenient reference when you want to find information quickly. It contains the following sections:

- *CodeWarrior IDE menu reference* on page D-2
- *CodeWarrior IDE default key bindings* on page D-17.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1 CodeWarrior IDE menu reference

This section gives an overview of the menu commands in the CodeWarrior IDE.

The following menus are always available:

- **File**
- **Edit**
- **View**
- **Search**
- **Project**
- **Debug** (this menu is not used by the CodeWarrior IDE for RVDS)
- **Window**
- **Help**.

The following menus are context-sensitive and are displayed only when a window that can use their menu commands is available:

- **Data** (this menu is not used by the CodeWarrior IDE for RVDS)
- **Browser**
- **Catalog** (this menu is not used by the CodeWarrior IDE for RVDS)
- **Layout** (this menu is not used by the CodeWarrior IDE for RVDS).

A **Version Control** menu is displayed if you have installed and configured the CodeWarrior IDE to work with a compatible version control system that you purchased separately. See the Metrowerks web site for information about the available VCS plug-ins for the CodeWarrior IDE.

The following sections describe the menus in the CodeWarrior IDE main menu bar:

- *File menu* on page D-3
- *Edit menu* on page D-5
- *View menu* on page D-7
- *Search menu* on page D-8
- *Project menu* on page D-10
- *Browser menu* on page D-13
- *Window menu* on page D-13
- *Help menu* on page D-14
- *View, Toolbars submenu* on page D-15
- *View, System submenu* on page D-16.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1.1 File menu

The **File** menu contains the commands you use to open, create, save, close, and print existing or new source code files and projects. The **File** menu also provides different methods for saving edited files. Table D-1 describes options from the **File** menu.

Table D-1 Menu options from the File menu

Menu option	Description
New...	This command opens the New dialog box which enables you to create new projects and files in the CodeWarrior IDE.
Open...	This command opens an existing file. See *Opening files from the File menu* on page 2-5 for more information.
Find and Open File...	This command opens an existing file, searching the current access paths as specified in the Access Paths panel of the Target Settings window. See *Opening header files from an editor window* on page 2-9 for more information.
Close	This command closes the active window. See *Closing files* on page 2-16 for more information.
Save	This command saves the contents of the active window to disk. See *Saving editor files* on page 2-12 for more information.
Save All	This command saves all editor files that are currently open. See *Saving all files* on page 2-13 for more information.
Save As...	This command saves the contents of the active window to disk under another name of your choosing. See *Renaming and saving a file* on page 2-13 for more information.
Save a Copy As...	This command saves the active window in a separate file. This command operates in different ways, depending on the active window. See *Saving a backup copy of a file* on page 2-14 for more information.
Revert...	This command reverts the active editor window to its last saved version. See *Reverting to the most recently saved version of a file* on page 2-20 for more information.
Open Workspace	This command opens a workspace `.cww` file. See *Opening workspaces* on page 3-22 for more information.
Close Workspace	This command closes the current workspace. See *Closing workspaces* on page 3-23 for more information.
Save Workspace	This commands saves the current workspace. See *Saving the current workspace or creating a new workspace* on page 3-22 for more information.
Save Workspace As...	This command creates a new workspace file from the current workspace. See *Saving the current workspace or creating a new workspace* on page 3-22 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-1 Menu options from the File menu (continued)

Menu option	Description
Import Components...	This menu item is not used by the CodeWarrior IDE for RVDS.
Close Catalog	This menu item is not used by the CodeWarrior IDE for RVDS.
Import Project...	This command imports an *eXtensible Markup Language* (XML) file into the CodeWarrior IDE so you can save the XML file as a CodeWarrior IDE project. The CodeWarrior IDE prompts you to choose a name and location to save the new project file. See *Importing and exporting a project as XML* on page 3-17 for more information.
Export Project...	This command exports a CodeWarrior IDE project to XML format. The CodeWarrior IDE prompts you to choose a name and location to save the new XML file. See *Importing and exporting a project as XML* on page 3-17 for more information.
Page Setup...	This command sets the options used when printing files from the CodeWarrior IDE. See *Setting print options* on page 2-18 for more information.
Print...	This command prints files from the CodeWarrior IDE on your printer. See *Printing a window* on page 2-18 or read the documentation that accompanies your printer for more information.
Open Recent	This command displays a submenu of projects and files that were recently opened. Select a filename from the submenu to open the file. If two or more files in the submenu have identical names, the full paths to those files are displayed to distinguish them. See *Opening files from the File menu* on page 2-5 for more information.
Exit	This command exits the CodeWarrior IDE immediately, provided either of the following conditions has been met: • all changes to the open editor files have already been saved • the open editor files have not been changed. If a project window is open, all changes to the project file are saved before the CodeWarrior IDE exits. If an editor window is open and changes have not been saved, the CodeWarrior IDE asks if you want to save the changes before exiting.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1.2 Edit menu

The **Edit** menu contains all the customary editing commands, along with some CodeWarrior IDE additions. This menu also includes the commands that open the Preferences and Target Settings windows. Table D-2 describes options from the **Edit** menu.

Table D-2 Menu options from the Edit menu

Menu option	Description
Undo	The text of this menu command varies depending on the most recent action, and your editor options settings. **Undo** reverses the effect of your last action. The name of the undo command varies depending on the type of operation you last executed. For example, if you have just typed in an open editor window, the undo command is renamed **Undo Typing**. Choosing the **Undo Typing** command removes the text you have just typed. See *Undoing the last edit* on page 6-17 and *Undoing and redoing multiple edits* on page 6-17 for more information. If you do not have **Use Multiple Undo** turned on in the Editor Settings preference panel, the **Undo** menu item toggles between Undo and Redo. See *Editor settings* on page 9-24 for more information.
Redo, Multiple Undo, and Multiple Redo	When an operation has been undone, it can be redone. For example, if you select **Undo Typing**, the menu item is changed to **Redo Typing**. Choosing this command overrides the previous undo. If you have **Use Multiple Undo** turned on in the Editor Settings preference panel, you have more flexibility to undo and redo operations. Select **Undo** multiple times to undo multiple actions. Select **Redo** multiple times to redo multiple actions. See *Undoing the last edit* on page 6-17 and *Undoing and redoing multiple edits* on page 6-17 for more information on undo and redo operations. See *Configuring other settings* on page 9-11 in the *Editor settings* on page 9-24 section for information on configuring multiple undo.
Cut	This command deletes the selected text and puts it in the system clipboard, replacing the contents of the clipboard.
Copy	This command copies the selected text in the active editor window onto the system clipboard. If the message window is active, the **Copy** command copies all the text in the message window onto the clipboard.
Paste	This command pastes the contents of the system clipboard into the active editor window. The **Paste** command replaces the selected text with the contents of the clipboard. If no text is selected, the clipboard contents are placed after the text insertion point. If the active window is the message window, the **Paste** menu item is grayed out and cannot be executed.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-2 Menu options from the Edit menu (continued)

Menu option	Description
Delete	This command deletes the selected text without placing it in the system clipboard. The **Delete** command is equivalent to pressing the Delete or Backspace key.
Select All	This command selects all the text in the active window. This command is usually used in conjunction with other **Edit** menu commands such as **Cut**, **Copy**, and **Clear**. See *Selecting text* on page 6-12 for more information.
Balance	This command selects the text enclosed in parentheses (), brackets [], or braces {}. For complete instructions on how to use this command, and how to balance while typing, see *Balancing punctuation* on page 6-14.
Shift Left	This command shifts the selected source code one tab size to the left. The tab size is specified in the Preferences window. See *Shifting text left and right* on page 6-16 for more information.
Shift Right	This command shifts the selected source code one tab size to the right. See *Shifting text left and right* on page 6-16 for more information.
Get Previous Completion	This command displays the previous suggested code completion item. See *Completing code* on page 6-15.
Get Next Completion	This command displays the next suggested code completion item. See *Completing code* on page 6-15.
Complete Code	This command displays the Code Completion window. See *Completing code* on page 6-15.
Preferences...	Use this command to change the global preferences for the CodeWarrior IDE. See *Choosing general preferences* on page 9-6 for more information.
Design Settings...	This menu item is not used by the CodeWarrior IDE for RVDS.
TargetName Settings...	Use this command to display the Target Settings window where you can change settings for the active build target. The name of this menu command varies depending on the name of your current build target. See *Configuring target settings* on page 10-8 for more information on the Target Settings window. See the menu item **Set Default Target** in Table D-5 on page D-10 for information on changing the current build target.
Version Control Settings...	This menu command displays the VCS settings window. (For more information about using the CodeWarrior IDE with a VCS, see the Metrowerks web site.)
Commands and Key Bindings...	Use this menu item to set key bindings for commands, and to customize the CodeWarrior IDE toolbars. See *Setting commands and key bindings* on page 9-37 for more information.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1.3 View menu

The **View** menu contains commands for viewing toolbars, browsers, and windows. Table D-3 describes options from the **View** menu.

Table D-3 Menu options from the View menu

Menu option	Description
Toolbars	This command causes the **Toolbar** submenu to appear. See *View, Toolbars submenu* on page D-15 for more information.
Component Catalog	This menu item is not used by the CodeWarrior IDE for RVDS.
Component Palette	This menu item is not used by the CodeWarrior IDE for RVDS.
Object Inspector	This menu item is not used by the CodeWarrior IDE for RVDS.
Project Inspector	This command enables you to view information about your project and enable debug information generation. See *Overview of the project window* on page 3-4 for more information.
Browser Contents	This command displays the browser Contents window. This menu command is grayed out when the browser is not activated. See *Viewing data by type with the Contents window* on page 8-17 for more information. See *Enabling browser data production* on page 8-5 for details of how to activate the browser.
Class Browser	This command displays the browser Class window. This menu command is grayed out when the browser is not activated. See *Viewing data by class with the Class browser window* on page 8-8 for more information. See *Enabling browser data production* on page 8-5 for details of how to activate the browser.
Class Hierarchy	This command displays the browser Hierarchy window. This menu command is grayed out when the browser is not activated. See *Viewing class hierarchies and inheritance with the hierarchy view* on page 8-18 for more information. See *Enabling browser data production* on page 8-5 for details of how to activate the browser.
Build Progress	This command displays the progress window for builds, as shown in Figure 4-16 on page 4-38.
Errors and Warnings	This command displays the Errors and Warnings window. See *Using the message window* on page 5-9 for more information. See also the Find All section in *Finding and replacing text with the Find and Replace dialog box* on page 7-4.
Symbolics	This menu item is not used by the CodeWarrior IDE for RVDS.
System	This command causes the **System** submenu to appear. Use this to monitor processes running in your system environment.
Breakpoints	This menu item is not used by the CodeWarrior IDE for RVDS.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-3 Menu options from the View menu (continued)

Menu option	Description
Registers	This window is not used by the CodeWarrior IDE for RVDS. See Chapter 5 *Working with the RealView Debugger* for more information.
Expressions	This window is not used by the CodeWarrior IDE for RVDS. See Chapter 5 *Working with the RealView Debugger* for more information.
Global Variables	This window is not used by the CodeWarrior IDE for RVDS. See Chapter 5 *Working with the RealView Debugger* for more information.
Command Window	Displays the CodeWarrior command window. The window provides a command-line interface to many of the CodeWarrior functions. See Appendix A *Running the CodeWarrior IDE from the Command Line and Command Window*.

D.1.4 Search menu

The **Search** menu contains all the necessary commands used to find text, replace text, and compare files. There are also some commands for code navigation. Table D-4 describes options from the **Search** menu.

Table D-4 Menu options from the Search menu

Menu option	Description
Find...	This command opens the Find dialog box which is used to find and/or replace the occurrences of a specific string in one or many files. See Chapter 7 *Searching and Replacing Text* for more information.
Replace...	This command replaces the selected text in the active window with the text string in the Replace text box of the Find window. If no text is selected in the active editor window, this command is grayed out. See *Finding and replacing text with the Find and Replace dialog box* on page 7-4 for more information. See *Selecting text* on page 6-12 for more information on selecting text.
Find in Files...	Opens the Find in Files window. Using this window, you can perform find-and-replace operations across a single file or multiple files. You can also specify various search criteria. See Chapter 7 *Searching and Replacing Text* for more information.
Find Next	This command finds the next occurrence of the Find text box string in the active window. This is an alternative to clicking the **Find** button in the Find dialog box. See *Finding and replacing text with the Find and Replace dialog box* on page 7-4 for more information.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-4 Menu options from the Search menu (continued)

Menu option	Description
Find in Next File	This command finds the next occurrence of the Find text box string in the next file listed in the Multi-File Search portion of the Find window (as exposed by the **Multi-File Search Disclosure triangle** in the Find window). This is an alternative to using the Find window. If the **Multi-File Search** button is not enabled this command is grayed out. See *Finding and replacing text in multiple files* on page 7-8 for more information.
Enter Find String	This command copies the selected text in the active window into the Find text box, making it the search target string. This is an alternative to copying text and pasting it into the Find window. See *Selecting text* on page 6-12 for more information.
Find Selection	This command finds the next occurrence of the selected text in the active text editor window. See *Finding and replacing text with the Find and Replace dialog box* on page 7-4 for more information.
Replace Selection	Substitutes the selected text in the active window with the text in the Replace field of the Find window. If no text is selected in the active editor window, the IDE grays out the menu command. Use the menu command to replace one instance of a text string without having to open the Find window. See Chapter 7 *Searching and Replacing Text* for information on finding and replacing text.
Replace and Find Next	This command replaces the selected text with the string in the Replace text box of the Find window, and then performs a **Find Next**. If no text is selected in the active editor window and there is no text in the Find text box string field of the Find window, this command is grayed out. See *Finding and replacing text with the Find and Replace dialog box* on page 7-4 for more information. See *Selecting text* on page 6-12 for more information on selecting text.
Replace All	This command finds all the occurrences of the Find string and replaces them with the Replace string. If no text is selected in the active editor window and there is no text in the Find text box in the Find dialog box, this command is grayed out.
Find Definition	This command searches for the definition of the function name selected in the active window. Searching occurs in the source files belonging to the open project. If the definition is found, the CodeWarrior IDE opens the source code file where the function is defined and highlights the function name. If the CodeWarrior IDE finds more than one definition, a message window appears warning you of multiple definitions. For more information on the message window, see *Using the message window* on page 5-13. If no definition is found, the system beeps.
Go Back	This command returns you to the previous view in the browser. See *Using Go Back and Go Forward* on page 8-21 for more information.
Go Forward	This command moves you to the next view in the browser (after you have used the **Go Back** command to return to a previous view). See *Using Go Back and Go Forward* on page 8-21 for more information.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-4 Menu options from the Search menu (continued)

Menu option	Description
Go to Line...	This command opens a Line Number dialog box (in which you enter a line number) and then moves the text insertion point to the line number you specify. See *Going to a specific line* on page 6-24 for more information.
Compare Files...	This command opens a **Compare Files Setup** dialog box to choose two files or directories to compare and merge. After choosing files to compare, a file comparison window appears, showing differences between the two files. If two directories are compared, the differences between the directories are shown in the **Compare Folders** window. See *Comparing and merging files and directories* on page 2-21 for more information.
Apply Difference	This command adds, removes, or changes text in the destination file shown in a file comparison window that is different from the text in the comparison window source file.
Unapply Difference	This command reverses the action of an **Apply Difference** command in a file comparison window.

D.1.5 Project menu

The **Project** menu enables you to add and remove files and libraries from your project. It also enables you to compile, build, and link your project. Table D-5 describes options from the **Project** menu.

Table D-5 Menu options from the Project menu

Menu option	Description
Add *window* to Project...	This command adds the file in the active editor window to the open project. See *Adding the current editor window* on page 3-29 for more information.
Add Files...	This command adds files to the project window. See *Using the Add Files command* on page 3-27 for more information.
Create Group...	The **Create Group...** command enables you to create a new group in the current project. This command is present in the **Project** menu if the Files category is selected in the current project window. See *Creating groups* on page 3-30 for more information.
Create Target...	The **Create Target** command enables you to create a new build target for the current project. This command is present in the **Project** menu if the **Targets** view is selected in the current project window. See *Working with multiple build targets and sub-projects* on page 3-40 for more information.
Create Segment/Overlay...	This menu item is not used by the CodeWarrior IDE for RVDS.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-5 Menu options from the Project menu (continued)

Menu option	Description
Create Design...	This menu item is not used by the CodeWarrior IDE for RVDS.
Export Project as GNU `Make` file	This command exports CodeWarrior IDE projects to GNU `Make` files. See *Converting GNU Make files* on page C-5.
Check Syntax	This command checks the syntax of the source code file in the active editor window or the selected file(s) in the open project window. If the active editor window is empty, or no project is open, this command is grayed out. **Check Syntax** does not generate object code. It only checks the source code for syntax errors. The progress of this operation is tracked in the toolbar message area. To abort this command at any time, press the Escape key. If one or more errors are detected, the message window appears. For information on how to correct compiler errors, consult *Correcting compilation errors and warnings* on page 5-14.
Preprocess	This command performs preprocessing on selected C and C++ source code files. See *Preprocessing source code* on page 4-40 for more information.
Precompile	This menu item is not used by the CodeWarrior IDE for RVDS.
Compile	This command compiles selected files. If the project window is active, the selected files and groups are compiled. If a source code file in an editor window is active, the source code file is compiled. The source code file must be in the open project. See *Compiling and linking a project* on page 4-35 for more information.
Compile If Dirty	This command compiles the file if it has been changed in a way that requires recompilation. **Note** This command does not appear in the **Project** menu by default, but you can add it using the Customize IDE Commands window. This is accessed using the **Commands and Key Bindings...** command in the **Edit** menu. See *Setting commands and key bindings* on page 9-37 for more information.
Disassemble	This command disassembles the compiled source code files selected in the project window, and displays object code in a new window. See *Disassembling code* on page 4-44 for more information.
Bring Up To Date	This command updates the open project by compiling all of its modified and touched files. See *Bringing a project up to date* on page 4-39 for more information.
Make	This command builds the selected project by compiling and linking the modified and touched files in the open project. The results of a successful build depend on the selected project type. See *Making a project* on page 4-41 for more information.
Stop Build	This command stops the current `Make` operation.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-5 Menu options from the Project menu (continued)

Menu option	Description
Remove Object Code...	This command removes all compiled source code binaries from the open project. The numbers in the **Code** column and **Data** column of each file are reset to zero. See *Removing objects from a project* on page 4-42 for more information.
Re-search for Files	To speed up builds and other project operations, the CodeWarrior IDE caches the locations of project files after it has found them in the access paths. The **Re-search for Files** option forces the CodeWarrior IDE to forget the cached locations of files and re-search for them in the access paths. This command is useful if you have moved files around on disk and want the CodeWarrior IDE to find them in their new locations. If the **Save project entries using relative paths** setting is enabled, the CodeWarrior IDE does not reset the relative path information stored with each project entry, so re-searching for files locates the source files in the same location. However, if the file no longer exists in the old location, the CodeWarrior IDE searches again, but only for header files. To force the CodeWarrior IDE to re-search for source files as well, you must first select **Reset Project Entry Paths**. If the **Save project entries using relative paths** setting is disabled, the CodeWarrior IDE re-searches for both header and source files.
Reset Project Entry Paths	This command resets the location information stored with each project entry when the **Save project entries using relative paths** setting is enabled. The next time the project entries are accessed, the CodeWarrior IDE searches for the project entries in the access paths. This command does nothing if the **Save project entries using relative paths** setting is disabled.
Synchronize Modification Dates	This command updates the modification dates stored in the project file. It checks the modification date for each file in the project, and if the file has been modified since it was last compiled, the CodeWarrior IDE marks it for recompilation. See *Synchronizing modification dates* on page 3-35 for more information.
Debug	This command compiles and links the current build target, and launches a debugger to debug the output image.
Run	This command compiles and links the current build target, and launches a debugger to run the output image. If the project type is set as a library, the **Run** command is grayed out.
Set Default Project	This menu command selects which project is the default project. See *Choosing a default project* on page 3-16 for more information.
Set Default Target	This command enables you to choose a different default target.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1.6 Browser menu

You can use the **Browser** menu to create new classes, member functions, and data members in the active project. This menu is added to the toolbar when a browser window is opened. Otherwise, the menu is not displayed. Table D-6 describes options from the **Browser** menu.

Table D-6 Menu options from the Browser menu

Menu option	Description
New Class...	This command displays a dialog box to help you create a new class for your project.
New Member Function...	This command displays a dialog box to help you create a new member function for a class in your project.
New Data Member...	This command displays a dialog box to help you create a new data member for a class in your project.
New Property..., New Method..., New Event Set..., New Event...	These menu items are not used by the CodeWarrior IDE for RVDS.

D.1.7 Window menu

The **Window** menu includes commands that tile open editor windows, switch between windows, and open Debugger windows. There is also a submenu for customizing the toolbars. Table D-7 describes options from the **Window** menu.

Table D-7 Menu options from the Window menu

Menu option	Description
Close	Closes the active window.
Close All *window*	Closes all open windows of a certain type. The name of this menu changes based on the type of item selected. For example, after you select one of several open editor windows, the menu command changes its name to **Close All Editor Documents**.
Cascade	This command opens all editor windows to their full screen size and stacks them one on top of another, with their window titles showing. This command is grayed out when the active window is the project window or the message window.
Tile Horizontally	Arranges open editor windows horizontally so that none overlaps another.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-7 Menu options from the Window menu (continued)

Menu option	Description
Tile Vertically	Arranges open editor windows vertically so that none overlaps another.
Save Default Window	Saves the settings of the active browser window. The CodeWarrior IDE applies the saved settings to subsequently opened browser windows.
Window list	The other **Window** menu items depend on which project, source files, header files, and other windows you have open. All the open windows are shown in this menu and the first nine files (1 through 9) are given key equivalents. The current project is always assigned the number 0 (zero). You must press Ctrl and a number to open a specific editor window. A check mark is placed beside the active window. To make one of your open CodeWarrior IDE files active and bring its window to the front, do one of the following: • click in its window • select it from the **Window** menu • use the key equivalent shown in the **Window** menu.

D.1.8 Help menu

Online Help is available from the **Help** menu.

Note

The Metrowerks Online Help system which is displayed from this menu does not include any information about the ARM customizations which have been made to CodeWarrior for RVDS. However, it does provide useful, general information about the features of CodeWarrior.

Table D-8 describes options from the **Help** menu.

Table D-8 Menu options from the Help menu

Menu option	Description
CodeWarrior Help	Displays the Metrowerks CodeWarrior Online Help system.
Index	Displays the Index window of the Metrowerks CodeWarrior Online Help system.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-8 Menu options from the Help menu (continued)

Menu option	Description
Search	Displays the Search window of the Metrowerks CodeWarrior Online Help system.
Metrowerks Website	Directs your web browser to the Metrowerks web site.
About Metrowerks CodeWarrior	Displays the About Metrowerks CodeWarrior IDE dialog box.

D.1.9 View, Toolbars submenu

The **View** menu contains a submenu for the **Toolbars** command. The **Toolbars** submenu contains all the commands used to customize the toolbars that appear in the CodeWarrior IDE windows. Table D-9 describes options from the **Toolbars** submenu.

Table D-9 Menu options from the Toolbars submenu

Menu option	Description
Show/Hide Window Toolbar	This command causes the CodeWarrior IDE to display (or hide) the toolbar in the active window. The command shown in the menu toggles between **Show Window Toolbar** and **Hide Window Toolbar**, depending on whether the toolbar of the active window is visible.
Reset Window Toolbar	This command causes the toolbar in the active window to reset to a default state. You can use this menu command if you want to return the editor window toolbar to its original default settings.
Clear Window Toolbar	This command causes the toolbar in the active editor, project, or browser window to have all icons removed from it. When all the icons have been removed, you can add icons using the **Toolbar Items** tab. Use the **Reset Window Toolbar** command to cause all the default icons to come back.
Show/Hide Main Toolbar	This command causes the CodeWarrior IDE to display (or hide) the main window toolbar. The command shown in the menu toggles between **Show Main Toolbar** and **Hide Main Toolbar**, depending on whether the toolbar is currently displayed or hidden.
Reset Main Toolbar	This command sets the main window toolbar to its default state.
Clear Main Toolbar	This command removes all elements from the main window toolbar. When all the icons have been removed, you can add icons using the **Toolbar Items** tab.

See *Customizing toolbars* on page 9-48 for more information on using this submenu.

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.1.10 View, System submenu

The **View** menu contains the **System** command. Table D-10 describes this command.

Table D-10 Menu option from the System submenu

Menu option	Description
Local Windows PC	This command causes the CodeWarrior IDE to display the System Browser - Local Windows PC window which displays all the current system processes running on the local PC.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.2 CodeWarrior IDE default key bindings

This section lists the default key bindings assigned to commands in the CodeWarrior IDE.

Some commands do not have any key bindings assigned to them by the CodeWarrior IDE. You can assign key bindings to any blank command. For more information on key bindings, see *Setting commands and key bindings* on page 9-37.

The commands in this section are listed in the order that they appear in the Windows layout:

- *File and project file commands*
- *Edit commands* on page D-18
- *View commands* on page D-19
- *Search commands* on page D-19
- *Project commands* on page D-20
- *Window commands* on page D-20
- *Miscellaneous commands* on page D-21
- *Editor commands* on page D-21.

Menu commands which are not shown in these sections do not have default key bindings assigned to them.

Note

Commands in the **Debug** menu do not apply to ARM.

D.2.1 File and project file commands

Table D-11 lists the default key bindings for the commands in the **File** menu.

Table D-11 File key bindings

Command	Key binding
New	Ctrl+N
New…	Ctrl+Shift+N
Open	Ctrl+O
Find and Open File	Ctrl+D
Find and Open File...	Ctrl+Shift+D

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-11 File key bindings (continued)

Command	Key binding
Close	Ctrl+W
Close All	Ctrl+Shift+W
Save	Ctrl+S
Save All	Ctrl+Shift+S
Print	Ctrl+P

D.2.2 Edit commands

Table D-12 contains the default key bindings for the commands in the **Edit** menu.

Table D-12 Edit key bindings

Command	Key binding
Undo	Ctrl+Z
Redo	Ctrl+Shift+Z
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Select All	Ctrl+A
Balance	Ctrl+B
Shift Left	Ctrl+[
Shift Right	Ctrl+]
Get Previous Completion	Alt+Shift+/
Get Next Completion	Alt+/
Complete Code	Alt+.
TargetName Settings	Alt+F7

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

D.2.3 View commands

Table D-13 contains the default key bindings for the commands in the **View** menu.

Table D-13 View key bindings

Command	Key binding
Class Browser	Alt+F12
Errors and Warnings	Ctrl+l

D.2.4 Search commands

Table D-14 contains the default key bindings for the commands in the **Search** menu.

Table D-14 Search key bindings

Command	Key binding
Find...	Ctrl+F
Replace...	Ctrl+H
Find In Files...	Ctrl+Shift+M
Find Next Find Previous	F3 Shift+F3
Find in Next File Find in Previous File	Ctrl+T Ctrl+Shift+T
Enter Find String	Ctrl+E
Find Selection	Ctrl+F3
Replace Selection	Ctrl+=
Replace & Find Next	Ctrl+L
Find Definition	Ctrl+’
Go Back	Ctrl+Shift+B
Go Forward	Ctrl+Shift+F
Goto Line	Ctrl+G

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-14 Search key bindings (continued)

Command	Key binding
Enter Replace String	Ctrl+Shift+E
Find Previous Selection	Ctrl+Shift+F3
Replace & Find Previous	Ctrl+Shift+L

D.2.5 Project commands

Table D-15 contains the default key bindings for commands in the **Project** menu.

Table D-15 Project key bindings

Command	Key binding
Check Syntax	Ctrl+;
Compile	Ctrl+F7
Disassemble	Ctrl+Shift+F7
Bring Up To Date	Ctrl+U
Make	F7
Stop Build	Ctrl+Break
Remove Object Code	Ctrl+–
Run/Debug	F5
Debug/Run	Ctrl+F5

D.2.6 Window commands

Table D-16 contains the default key bindings for the commands in the **Window** menu.

Table D-16 Window menu key bindings

Command	Key binding
Close	Ctrl+W

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-16 Window menu key bindings (continued)

Command	Key binding
Close All	Ctrl+Shift+W
Select Default Project	Ctrl+0
Select Document *n*	Ctrl+*n*

D.2.7 Miscellaneous commands

Table D-17 contains the default key bindings for handling miscellaneous tasks in the CodeWarrior IDE.

Table D-17 Miscellaneous key bindings

Command	Key binding
Go to Header/Source File	Ctrl+`
Go to Previous Error Message	F4
Go to Next Error Message	Shift+F4

D.2.8 Editor commands

Table D-18 contains the default key bindings for handling editor windows in the CodeWarrior IDE.

Table D-18 Editor window key bindings

Command	Key binding
Move Character Left	Left Arrow
Move Character Right	Right Arrow
Move Word Left	Ctrl+Left Arrow
Move Word Right	Ctrl+Right Arrow
Move Sub-word Left	Alt+Left Arrow
Move Sub-word Right	Alt+Right Arrow
Move to Start of Line	Home

 Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-18 Editor window key bindings (continued)

Command	Key binding
Move to End of Line	End
Move Line Up	Up Arrow
Move Line Down	Down Arrow
Move to Top of Page	Page Up
Move to Bottom of Page	Page Down
Move to Top of File	Ctrl+Home
Move to Bottom of File	Ctrl+End
Delete Character Left	Backspace
Delete Character Right	Del
Character Select Left	Shift+Left Arrow
Character Select Right	Shift+Right Arrow
Select Word Left	Ctrl+Shift+Left Arrow
Select Word Right	Ctrl+Shift+Right Arrow
Select Sub-word Left	Alt+Shift+Left Arrow
Select Sub-word Right	Alt+Shift+Right Arrow
Select Line Up	Shift+Up Arrow
Select Line Down	Shift+Down Arrow
Select to Start of Line	Shift+Home
Select to End of Line	Shift+End
Select to Start of Page	Shift+Page Up
Select to End of Page	Shift+Page Down
Select to Start of File	Ctrl+Shift+Home
Select to End of File	Ctrl+Shift+End
Scroll Line Up	Ctrl+Up Arrow
Scroll Line Down	Ctrl+Down Arrow

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Table D-18 Editor window key bindings (continued)

Command	Key binding
Find Symbols with Prefix	Ctrl+\
Find Symbols with Substring	Ctrl+Shift+\
Get Next Symbol	Ctrl+.
Get Previous Symbol	Ctrl+,
Add bookmark	Ctrl+F2
Go to next bookmark	F2
Go to previous bookmark	Shift+F2

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.

Copyright © 1999-2001, 2005-2007 ARM Limited. All rights reserved.