RealView Developer Suite

Version 2.2

CodeWarrior® IDE Guide

ARM DUI 0065E

RealView Developer Suite
CodeWarrior IDE Guide

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Change
October 1999	A	Release 1.0 (ADS)
March 2000	B	Release 1.0.1 (ADS)
November 2000	C	Release 1.1 (ADS)
November 2001	D	Release 1.2 (ADS)
May 2005	E	RVDS 2.2 Release

Proprietary Notice

Confidentiality Status

Product Status

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

Contents
RealView Developer Suite CodeWarrior IDE Guide

Preface

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

- *About this book* on page viii
- *Feedback* on page xii.

About this book

This book provides user information for *the CodeWarrior IDE for the RealView Developer Suite*. It describes the major graphical user interface components of the CodeWarrior IDE, and provides information on ARM-specific features.

Intended audience

This book is written for all developers who are using the ARM version of the CodeWarrior IDE to manage their ARM-targeted RealView development projects under all current versions of the Windows OS. 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 Developer 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.

Chapter 5 ***Working with the ARM Debuggers***

Read this chapter for details of how to use the ARM debuggers with the CodeWarrior IDE. It describes how the CodeWarrior IDE interacts with the ARM debuggers. It also describes parts of the CodeWarrior IDE that are useful for finding errors in your code, such as the CodeWarrior IDE 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 CodeWarrior IDE 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. In addition it gives information on configuring command keybinding 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 compiler, assembler, debuggers, `fromelf`, and other tools, to produce machine code for execution on an ARM processor.

It also describes how to configure important target-specific options for the CodeWarrior IDE, such which linker and post-linker to use, and the access paths and file mappings that apply to a build target.

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.

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 which can be 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 which can be used to import projects created by other IDE applications.

Appendix D ***CodeWarrior IDE Reference***

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

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
<u>`mono`</u>`space`	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.

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 periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list in the Technical Support area of the ARM web site at `http://www.arm.com`.

ARM publications

This book contains information that is specific to the version of the CodeWarrior IDE supplied with the *RealView Developer Suite* (RVDS). Refer to the following books in the RVDS document suite for information on other components of RVDS:

- *RealView Developer Suite v2.2 Getting Started Guide (ARM DUI 0255)*
- *RealView Compilation Tools v2.2 Developer Guide (ARM DUI 0203)*
- *RealView Compilation Tools v2.2 Assembler Guide (ARM DUI 0204)*
- *RealView Compilation Tools v2.2 Compiler and Libraries Guide (ARM DUI 0205)*
- *RealView Compilation Tools v2.2 Linker and Utilities Guide (ARM DUI 0206)*
- *RealView Debugger Essentials Guide (ARM DUI 0181)*
- *RealView Developer Suite AXD and armsd Debuggers Guide (ARM DUI 0066)*

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 books are referenced in the text:

Friedl, J., *Mastering Regular Expressions*, 1997, O'Reilly & Associates, International Thomson Publishing. ISBN 1565922573.

Feedback

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

Feedback on the RealView Developer Suite

If you have any problems with the RealView Developer Suite, please contact your supplier. To help them provide a rapid and useful response, please give:

- 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 stand-alone 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 tool, including the version number and date.

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.

Chapter 1
Introduction

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

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 RVDS to develop C, C++, and ARM assembly language code targeted at ARM and Thumb 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

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

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

Target system	The specific ARM-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 of code targeted at hardware based on an ARM7TDMI® processor. Build targets can share files in the same project while using their own settings.

The CodeWarrior IDE provides:

- 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

- 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.

1.2 About the CodeWarrior IDE for the RealView Developer Suite

The CodeWarrior IDE for the RealView Developer Suite is based on Metrowerks CodeWarrior IDE version 5.6.1. A CodeWarrior plug-in has been developed by ARM to provide support for the ARM RealView Developer Suite (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.

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. In most cases, these are related to debugging, because the ARM debuggers are provided separately. In particular:

- There are a number of configuration dialogs that are not used by the RVDS toolchain, such as the **Runtime Settings** target configuration dialog. See Chapter 10 *Configuring a Build Target* for more information on target configuration.
- The ARM debuggers are not tightly integrated with the CodeWarrior IDE. This means, for example, that you cannot set breakpoints or watchpoints from within the CodeWarrior IDE. Instead you must use one of the following ARM debuggers supplied with RVDS:
  - *RealView Debugger* (RVD)
  - *ARM eXtended Debugger* (AXD)
  - *ARM Symbolic Debugger* (`armsd`)

  See *How the ARM debuggers 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.

Interface items that are not used by the CodeWarrior IDE for RVDS are documented as *Not used by the CodeWarrior IDE for RVDS* in the documentation and the online help.

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.

1.2.1 Unused menu commands and windows

The following commands and windows are not implemented in the ARM version of the CodeWarrior IDE. (For a more detailed list of the menu commands, refer to Appendix D *CodeWarrior IDE Reference*.):

File menu

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

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

Edit menu

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

- **Design Settings...**

View menu

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

- **Component Catalog**
- **Component Palette**
- **Object Inspector**
- **Symbolics**
- **Breakpoints**
- **Registers**
- **Expressions**
- **Global Variables**

Project menu

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

- **Create Design...**
- **Precompile**. (The RealView compiler does not support precompiled headers.)

Debug menu

None of the menu commands in the **Debug** menu are applicable to ARM. See Chapter 5 *Working with the ARM Debuggers* for more information.

Tools menu

None of the menu commands in the **Tools** menu are applicable to ARM.

Data menu

None of the menu commands in the **Data** menu are applicable to ARM.

Browser menu

The following **Browser** menu items are not used:

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

Catalog menu

None of the menu commands in the **Catalog** menu are applicable to ARM.

Layout menu

None of the menu commands in the **Layout** menu are applicable to ARM.

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.
- GNU Makefiles can be directly imported into CodeWarrior for RVDS. CodeWarrior projects can also be exported as GNU Makefiles.
- XML files (formatted in CodeWarrior for RVDS XML format) can be directly imported into CodeWarrior for RVDS.
- RealView Debugger (RVD) project files must be saved to XML format before they can be imported into CodeWarrior for RVDS.

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

1.3 Where to go from here

The following documentation and examples will help you get started with the CodeWarrior IDE:

- Read the *RealView Developer Suite Getting Started Guide* for a quick introduction to the CodeWarrior IDE for RVDS.
- Examine the example ARM projects provided in the `examples` subdirectory of your RVDS installation directory.
- See Chapter 3 *Working with Projects* for introductory information about CodeWarrior IDE projects.
- See Chapter 4 *Working with the ARM Project Stationery* for information on using the ARM project stationery provided with the CodeWarrior IDE for RVDS.

  ——— **Note** ———

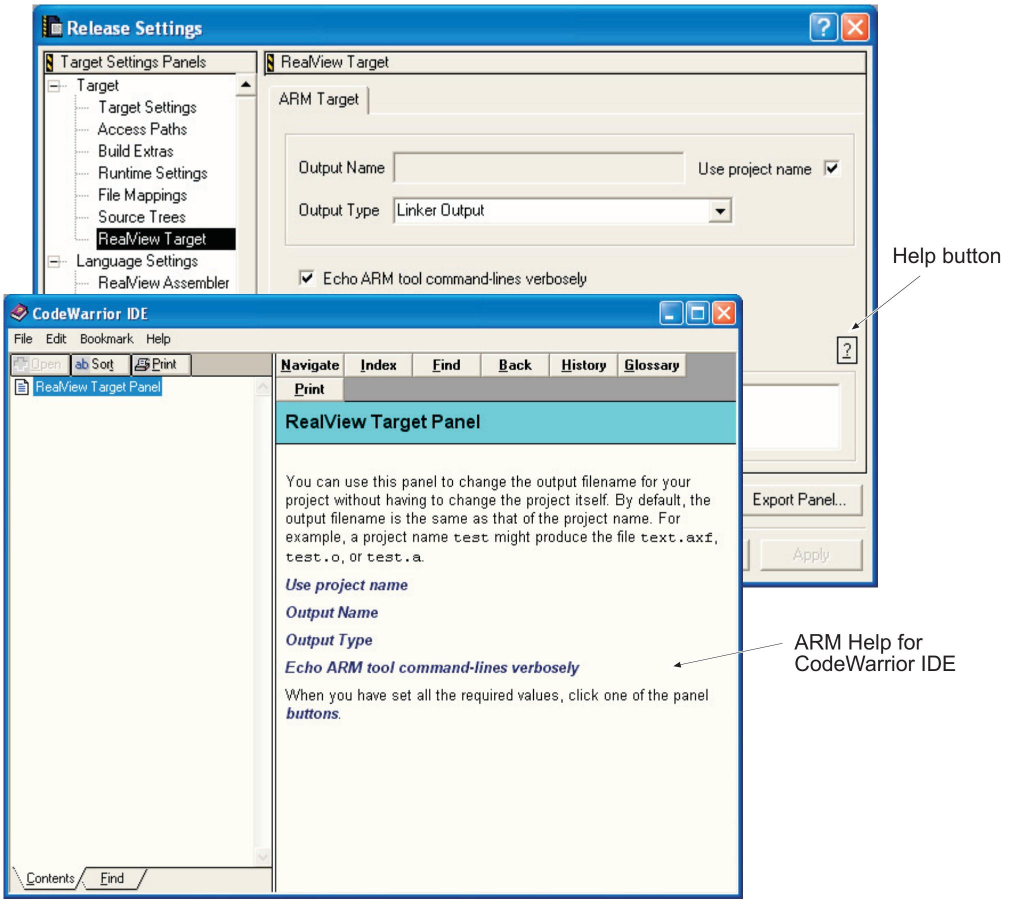
  If you are setting up a complex project environment, refer to *Configuring the CodeWarrior IDE for complex or multi-user projects* on page 3-42 for important information.

  ————————
- See Chapter 10 *Configuring a Build Target* for information on configuring the RVDS toolchain from within the CodeWarrior IDE.
- See Appendix C *CodeWarrior Project Conversion Utilities* for information on converting projects or makefiles created using other development environments.

1.3.1 Online documentation and online help

Documentation for the CodeWarrior IDE for RVDS is available online as part of the RVDS documentation collection. Select **Programs** → **ARM** → **DynaText Documentation** from the Windows **Start** menu to access the collection. The CodeWarrior IDE guide is available from the **RealView Developer Suite v2.2** collection.

In addition, the CodeWarrior IDE for RVDS provides context-sensitive online help for those areas of CodeWarrior that have been customized by ARM. You can access this Help information by clicking on the Help button on the panel. See Figure 1-1:



RealView Target Panel

You can use this panel to change the output filename for your project without having to change the project itself. By default, the output filename is the same as that of the project name. For example, a project name `test` might produce the file `text.axf`, `test.o`, or `test.a`.

Use project name

Output Name

Output Type

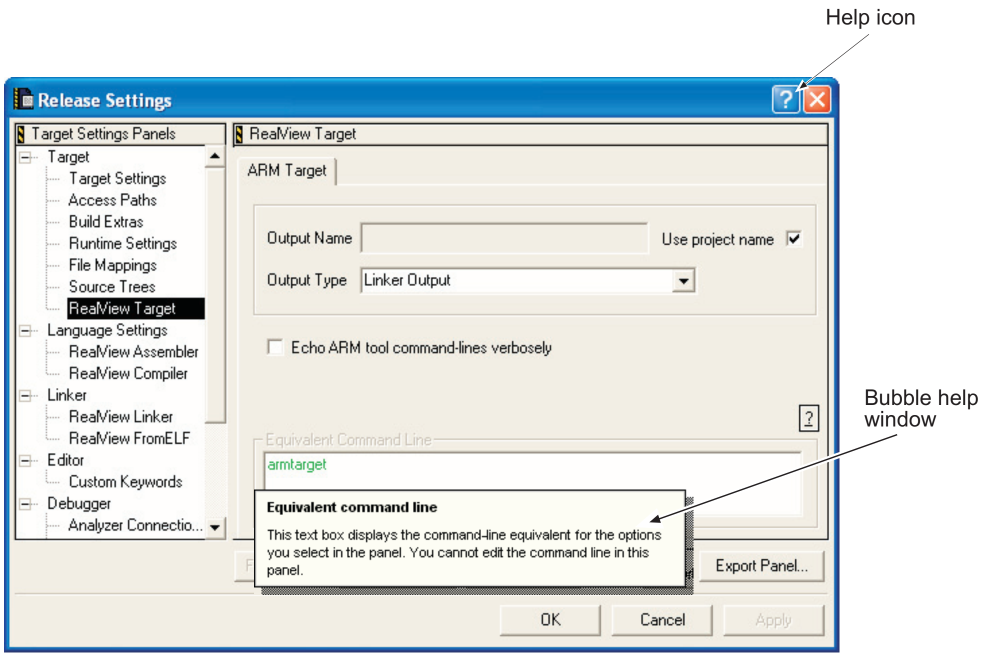
Echo ARM tool command-lines verbosely

When you have set all the required values, click one of the panel *buttons*.

Figure 1-1 Context-sensitive Help for CodeWarrior panels customized by ARM

The Help button only appears on Panels which have been customized by ARM. If there is no Help button on the Panel, then this Panel has not been customized by ARM.

All CodeWarrior panels provide a Help icon in the top-right corner of a Panel which displays abbreviated 'bubble-help' information for all items in the panel. To display help information, click on the Help icon and then click on an item in the Panel. See Figure 1-2:



Output Name:
Use project name ✓
Output Type: Linker Output
☐ Echo ARM tool command-lines verbosely
Equivalent Command Line: armtarget

Equivalent command line

This text box displays the command-line equivalent for the options you select in the panel. You cannot edit the command line in this panel.

Figure 1-2 Bubble-help for CodeWarrior panels

Note

There is also CodeWarrior Online Help provided from the **Help** menu or by pressing the **F1** key. This Online Help provides general information about CodeWarrior but does not include any information about the areas of CodeWarrior which have been customized by ARM.

Chapter 2
Working with Files

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

2.1 About working with files

This chapter gives information on 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 files.

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

- 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 folders* 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.

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. Although, you can add libraries to a project and perform disassembly on them.

———————————

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 (Figure 2-1 on page 2-4).

   ——— **Note** ———

   See *Creating a new project* on page 3-13 for more information on the **Project** tab. The **Component Catalog File** entry in this tab and the **Object** tab are not used by the ARM version of the CodeWarrior IDE.

   ———————————

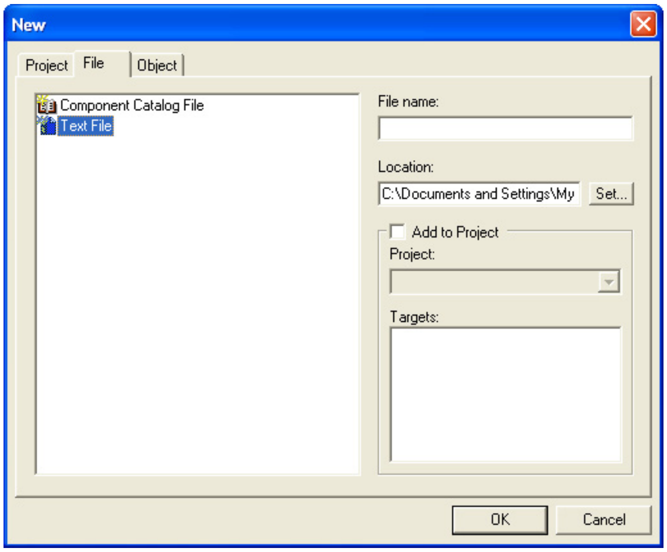


Figure 2-1 The New dialog box

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

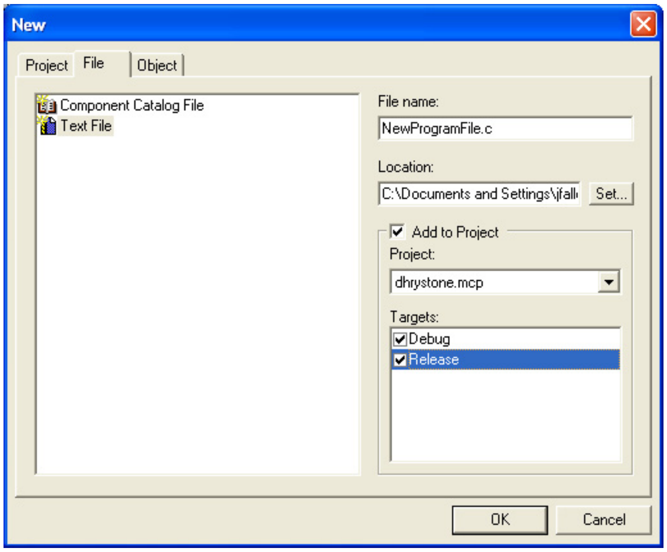
3. Enter a file name for the new file in the **File name** text field. If you want to add the new file to a build target in an existing project, you must ensure that the filename you enter uses a filename extension that is defined in the File mappings panel for the build target. See *Configuring file mappings* on page 10-24 for more information.

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

5. Select the **Add to Project** checkbox if you want to add the new file to an existing project (Figure 2-2 on page 2-5):

   a. Click the **Project** drop-down list to select the project you want to add the file to from the pop-up list of currently open projects. 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.



File name: NewProgramFile.c

Location: C:\Documents and Settings\jfall

☑ Add to Project

Project: dhrystone.mcp

Targets:
☑ Debug
☑ Release

Figure 2-2 Add new file to project

6. Click **OK** to create the new file. If you have selected **Add to Project**, the new file is added to the selected build targets, provided the filename extension of the new file is defined in the File mappings configuration panel for the 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 choose not to associate the file with a project, you can do so later. See *Adding files to a project* on page 3-29 for more information.

2.2.2 Opening files from the File menu

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

- Project files. See *Working with simple projects* on page 3-13 for information on opening projects.
- Text files, such as a source code file, header file, or other text file.

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).

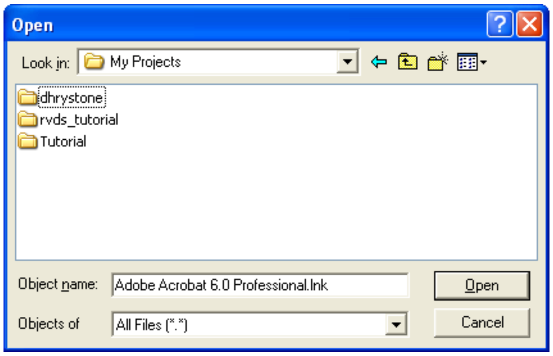


Figure 2-3 Open dialog box

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 within 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.

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 3-28 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. To view the contents of library files, right click on the library file and select **Disassemble** from the context menu.

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 within the group. Figure 2-4 on page 2-8 shows an example.

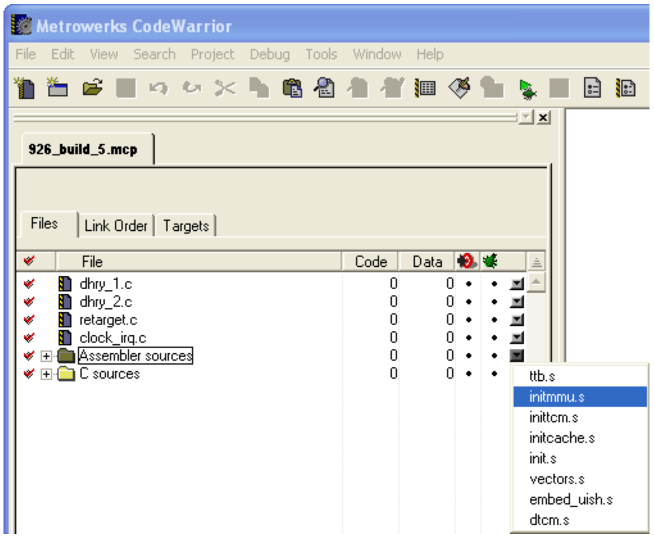


Figure 2-4 Group drop-down menu

3. Select the menu item for the source file that you want to open. The file is opened in an editor window.

See *Grouping files in a project* on page 3-33 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:

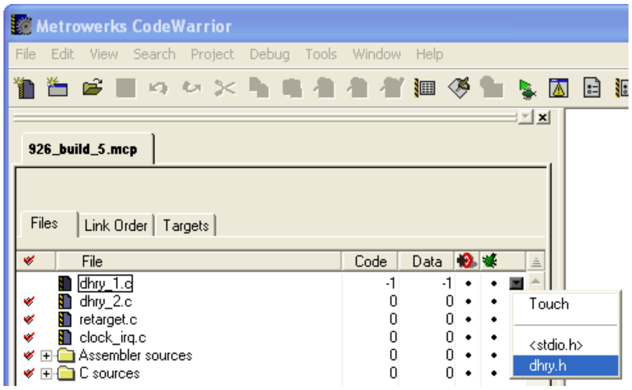
1. Select the source file in the project window.

2. Click the **Header Files** pop-up button. A list of header files is displayed. Figure 2-5 on page 2-9 shows an example.

   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 that are 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.



	File	Code	Data
	dhry_1.c	-1	-1
✓	dhry_2.c	0	0
✓	retarget.c	0	0
✓	clock_irq.c	0	0
✓	Assembler sources	0	0
✓	C sources	0	0

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

- You can 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** pop-up button for a library file that is part of your project, you will 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 3-38 for more information on touching files.

2.2.4 Opening header files from an editor window

The following sections describe various ways to open header files from within 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 within a source file you are editing:

1. Click the **Header Files** drop-down menu at the top left of the editor window (see Figure 6-15 on page 6-25). 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.

   Note

   If there are no files available in the drop-down menu, it means one or more 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 any of the following methods:
     - select **Find and Open File** from the **File** menu
     - right-click and choose **Find and Open File** from the menu displayed
     - 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 dialog to search for the header file.
- 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 any of the following methods:
     - select **Find and Open File** from the **File** menu
     - right-click and choose **Find and Open File** from the menu displayed
     - press Ctrl-d.

The CodeWarrior IDE displays the **Find and Open File** dialog box (Figure 2-6):

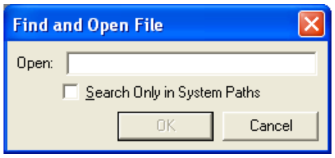


Figure 2-6 Find and open file dialog box

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 dialog to search for header files.
3. Select the **Search Only in System Paths** option if you want to restrict the search to the directories specified in the System Paths pane of the Access Paths configuration dialog.

   Turn the **Search Only in System Paths** option off to search both System Paths pane and User Paths pane directory paths (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 back and forth 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 settings panel to find the header file. See *Configuring access paths* on page 10-12 for more information on configuring access paths.

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 need 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 is an existing file, 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.

———————————

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 editor saves all the modified files to your hard disk.

Saving files automatically

The CodeWarrior IDE can automatically save changes to all your modified files whenever you select any of the following menu 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, you can turn this option off.

Caution

The ARM debuggers read source files from disk. If you are debugging at source level and select this option, the debuggers will 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 (Figure 2-7 on page 2-14).

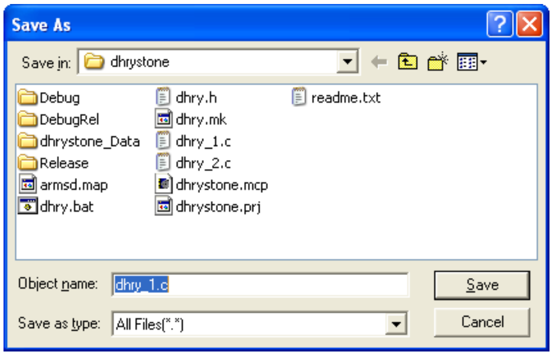


Figure 2-7 Save As dialog box

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 altering the original format of the file.

See *Specifying Other Settings* on page 9-26 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.

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

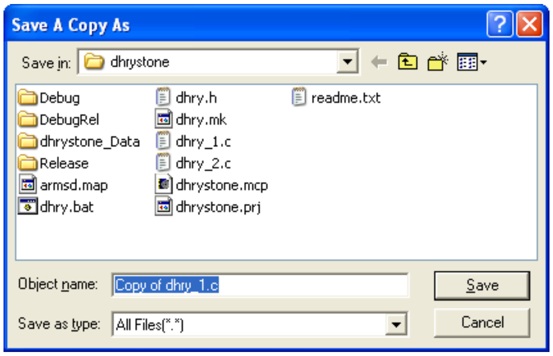


Figure 2-8 Save A Copy As dialog box

3. Type the name and location for the new file in the File 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 2-29 for information on exporting a project to XML.

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 choose **Close** from the menu displayed.)
- click the Windows 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 3-18 for more information on saving project files.

2.4.2 Closing editor files

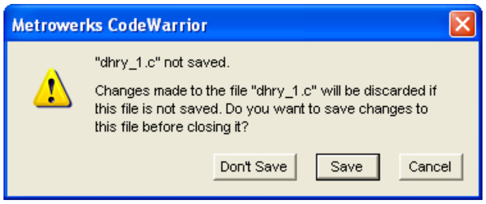
This section describes how to close editor files.

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 (Figure 2-9 on page 2-17).


"dhry_1.c" not saved.

Changes made to the file "dhry_1.c" will be discarded if this file is not saved. Do you want to save changes to this file before closing it?

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 9-23 for information on how to configure these options. If the appropriate options are enabled, the editor window will occupy the same position on your screen and will have the same text selected the next time the source code file is opened.

————————————

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 printer 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.

In addition, there are four CodeWarrior IDE-specific print options:

Print Selection Only

This option is available only 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.

Print using 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

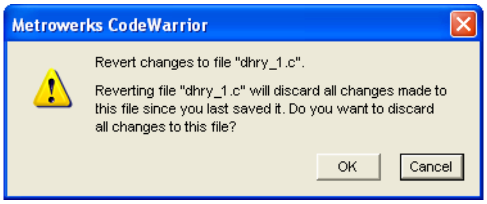
Select this option to print line numbers alongside each line of the file.

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

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



Revert changes to file "dhry_1.c".

Reverting file "dhry_1.c" will discard all changes made to this file since you last saved it. Do you want to discard all changes to this file?

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 9-26 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. Refer to the Metrowerks web site for information about the available VCS plug-ins for the CodeWarrior IDE.

2.7 Comparing and merging files and folders

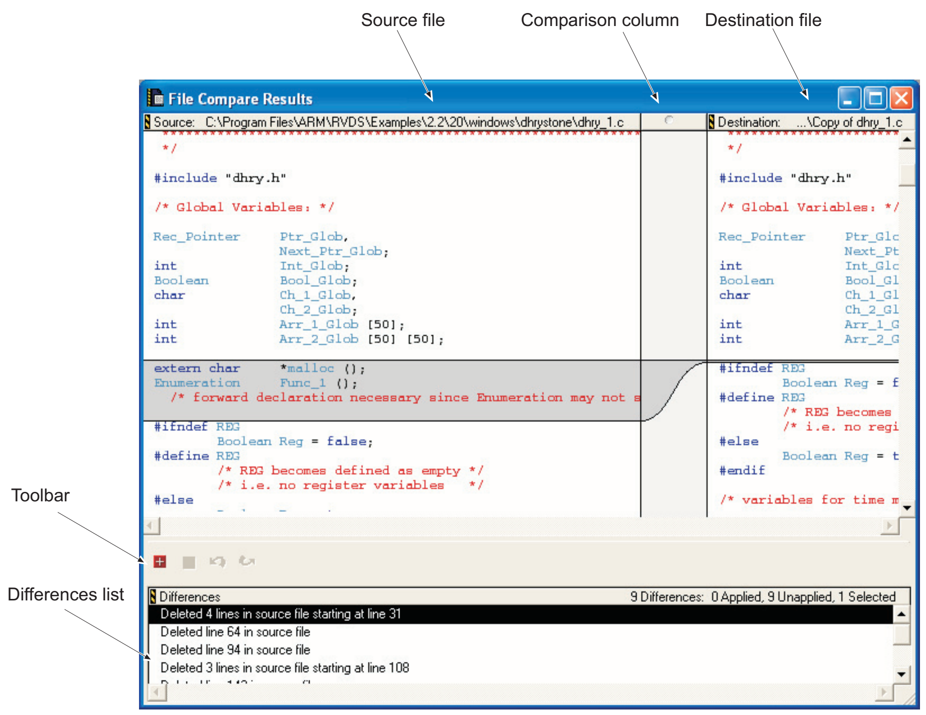
You can use the CodeWarrior IDE to compare two text files, mark the differences between the files, and apply changes to the files. In addition, you can compare the contents of two folders.

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 2-24
- *Applying and unapplying differences* on page 2-25
- *Choosing folders to compare* on page 2-26
- *Comparing XML-formatted projects* on page 2-29.

2.7.1 File comparison and merge overview

The file comparison window displays two text files, and the differences between them. 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. Figure 2-11 on page 2-22 shows an example.



Source: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone\dhry_1.c

```
 */

#include "dhry.h"

/* Global Variables: */

Rec_Pointer     Ptr_Glob,
                Next_Ptr_Glob;
int             Int_Glob;
Boolean         Bool_Glob;
char            Ch_1_Glob,
                Ch_2_Glob;

int             Arr_1_Glob [50];
int             Arr_2_Glob [50] [50];

extern char     *malloc ();
Enumeration     Func_1 ();
  /* forward declaration necessary since Enumeration may not s
#ifndef REG
        Boolean Reg = false;
#define REG
        /* REG becomes defined as empty */
        /* i.e. no register variables   */
#else
```

Destination: ...\Copy of dhry_1.c

```
 */

#include "dhry.h"

/* Global Variables: */

Rec_Pointer     Ptr_Glc
                Next_Pt
int             Int_Glc
Boolean         Bool_Gl
char            Ch_1_Gl
                Ch_2_Gl

int             Arr_1_G
int             Arr_2_G

#ifndef REG
        Boolean Reg = f
#define REG
        /* REG becomes
        /* i.e. no regi
#else
        Boolean Reg = t
#endif

/* variables for time m
```

Differences — 9 Differences: 0 Applied, 9 Unapplied, 1 Selected

```
Deleted 4 lines in source file starting at line 31
Deleted line 64 in source file
Deleted line 94 in source file
Deleted 3 lines in source file starting at line 108
```

Figure 2-11 The File Compare Results window

The main parts of the **File Compare Results** window are:

Source file column

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

Destination file column

This column displays the destination file that is compared with the source file. The destination file is displayed on the right side of the file comparison window. 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 displays a graphical representation of where text was added or removed between the source and destination files. This column is displayed between the source and destination panes in the comparison window.

Differences list

The Differences list lists 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 panes. Text in the Differences list is displayed in italics when a difference is applied to the destination file.

Toolbar

The toolbar has buttons to apply or remove changes between the two files to the destination file. The toolbar also has buttons to undo and redo changes to the source and destination files. See *Customizing toolbars* on page 9-48 for information on how to customize the toolbar.

Table 2-1 shows the control icons.

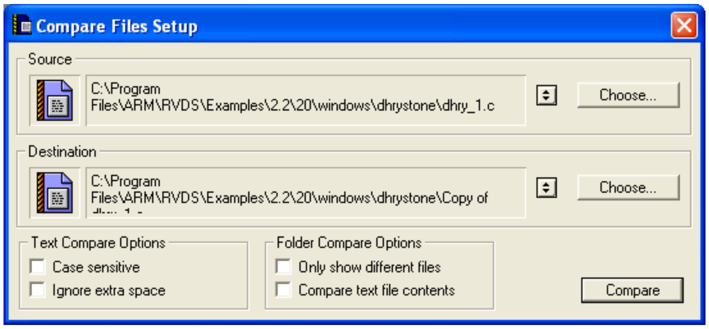
Table 2-1 File Compare Toolbar control icons

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).



Source
C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone\dhry_1.c

Destination
C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone\Copy of

Text Compare Options
☐ Case sensitive
☐ Ignore extra space

Folder Compare Options
☐ Only show different files
☐ Compare text file contents

Figure 2-12 Compare Files Setup window

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 alongside the **Choose** button. A menu is displayed showing all the files which are currently open. Choose a file to compare from the menu.

3. Set the text compare options you want. The available options are:

   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 folders 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.

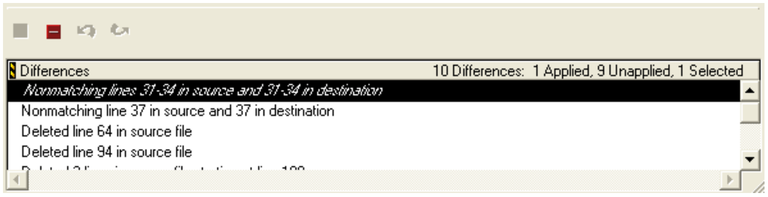
2.7.3 Applying and unapplying differences

Use the Comparison window toolbar and Differences list window to select the differences that you want to apply from the source file to the destination file.

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. See Figure 2-13:



Differences 10 Differences: 1 Applied, 9 Unapplied, 1 Selected

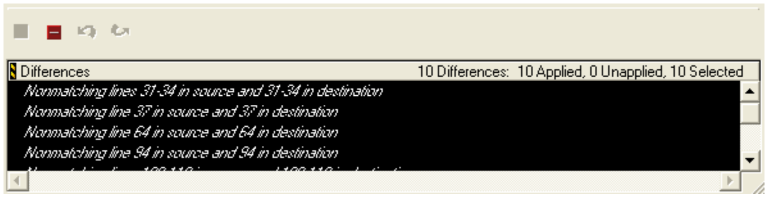
Nonmatching lines 31-34 in source and 31-34 in destination
Nonmatching line 37 in source and 37 in destination
Deleted line 64 in source file
Deleted line 94 in source file

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. Choose **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 Difference list. See Figure 2-14 on page 2-26:


Differences   10 Differences: 10 Applied, 0 Unapplied, 10 Selected

Nonmatching lines 31-34 in source and 31-34 in destination
Nonmatching line 37 in source and 37 in destination
Nonmatching line 64 in source and 64 in destination
Nonmatching line 94 in source and 94 in destination

Figure 2-14 Applied all differences

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.

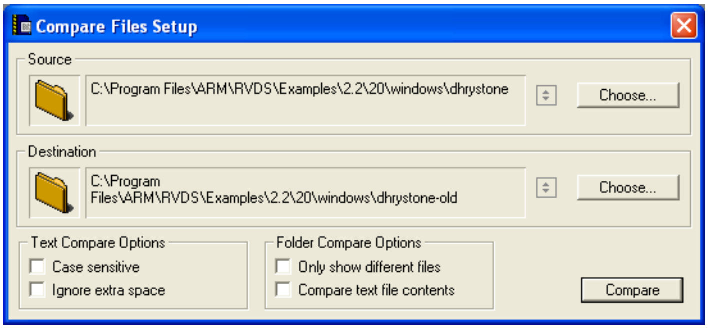
Note

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 folders to compare

You can use the CodeWarrior IDE comparison features to compare complete folders of files. To compare two folders:

1. Select **Compare Files** from the **Search** menu. The CodeWarrior IDE displays a Compare Files Setup dialog that prompts you for a source folder and destination folder to compare (Figure 2-15 on page 2-27).



Compare Files Setup

Source

C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone

Destination

C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone-old

Text Compare Options

- [ ] Case sensitive
- [ ] Ignore extra space

Folder Compare Options

- [ ] Only show different files
- [ ] Compare text file contents

Figure 2-15 Compare Folders Setup dialog box

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

   Alternatively, you can drag and drop folders from the Windows desktop to the source and destination sections of the window.

3. Set the folder comparison options you want. The available options are:

   Only show different files

   Select this option to display only files that are different in both folders in the Files In Both Folders list of the Compare Folders window (Figure 2-16 on page 2-28). By default, this option is disabled, so all files in the source and destination folders are displayed.

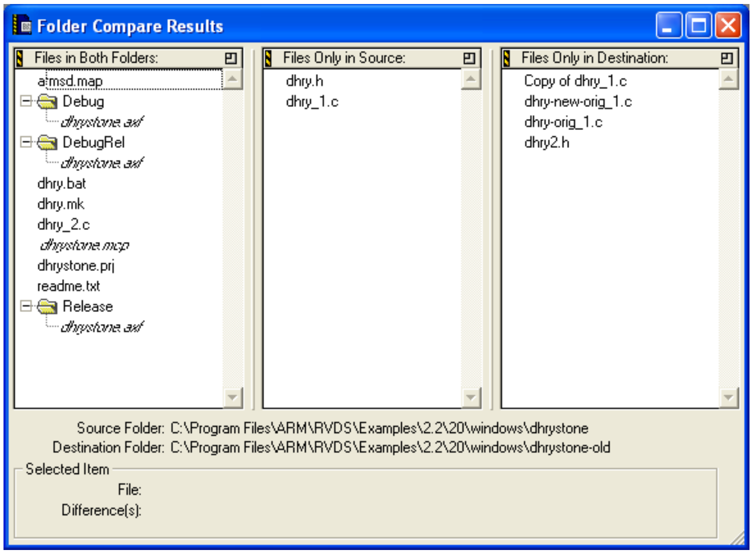
   Comparisons between files in the source and destination folders are normally based on the file modification dates and file sizes. This is usually good enough to determine if there are differences between the two files. If there are invisible items in the folders, the comparison will skip over those items.

   Compare text file contents

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

   See *Choosing files to compare* on page 2-24 for information on the file comparison options.

4. Click **Compare** to compare the two folders. The CodeWarrior IDE displays the Folder Compare Results window (Figure 2-16 on page 2-28). The names of source code files, header files, text files, and folders are displayed in plain face. All other file names are displayed in italics.



Folder Compare Results

Files in Both Folders:
- armsd.map
- Debug
  - dhrystone.axf
- DebugRel
  - dhrystone.axf
- dhry.bat
- dhry.mk
- dhry_2.c
- dhrystone.mcp
- dhrystone.prj
- readme.txt
- Release
  - dhrystone.axf

Files Only in Source:
- dhry.h
- dhry_1.c

Files Only in Destination:
- Copy of dhry_1.c
- dhry-new-orig_1.c
- dhry-orig_1.c
- dhry2.h

Source Folder: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone
Destination Folder: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone-old

Selected Item
File:
Difference(s):

Figure 2-16 Folder Compare Results window

The files are displayed in three lists:

Files in Both Folders

This list displays all files in both the source and destination folders, unless the **Only Show Different Files** option is enabled. Files that are different in the two folders 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 folder.

Files Only in Destination

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

5. Click on a file name in any of the lists to display specific information about the selected file in the **Selected Item** box at the bottom of the folder comparison 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

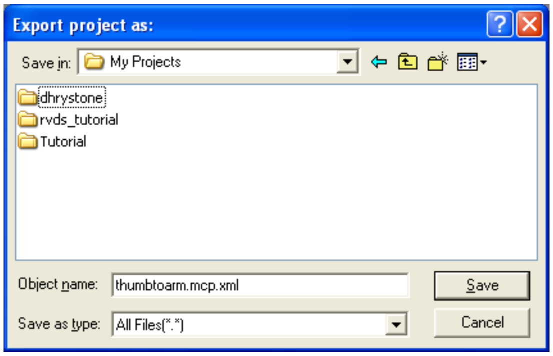
You can click on a zoom box (◰) for any of the three lists to expand them to fill the window. Click the zoom box again to collapse the lists back to their original size.

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 properly reflect the differences in the two files.
   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).



Export project as:

Save in: My Projects

dhrystone
rvds_tutorial
Tutorial

Object name: thumbtoarm.mcp.xml

Save as type: All Files(*.*)

Figure 2-17 Export project as dialog box

   d. Save the project as an XML file.
   e. Repeat steps a to d 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. Choose the XML-formatted versions of the two project files in the Compare Files Setup dialog box. See *Choosing files to compare* on page 2-24 if you do not know 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 2-25 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 asking you to name the project file that will be created, and choose 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 refer to *Converting RVD project files* on page C-9.

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:

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:

- the ARM-specific project stationery provided with this version of the CodeWarrior IDE
- how you can 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 that are 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.

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 3-52 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

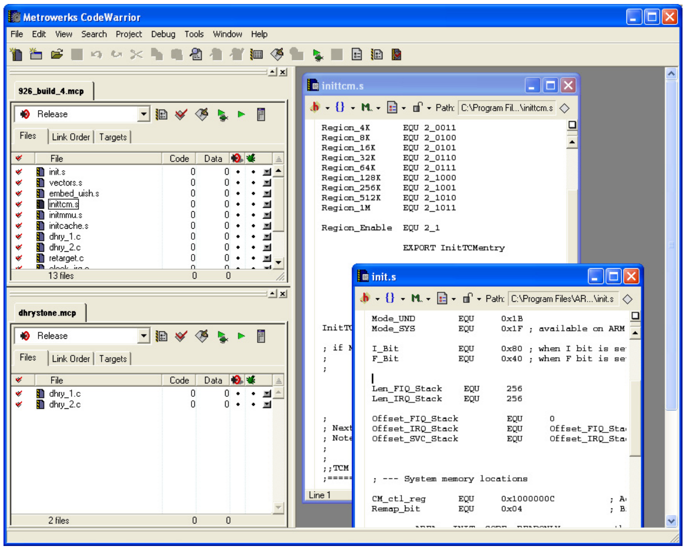
You can use the output from one build target as input to another, or create independent build targets that generate different types of output. You can use dependent build targets, for example, to combine output objects from ARM and Thumb build targets into a single output image. See *Working with multiple build targets and subprojects* on page 3-44 for detailed information on build targets.

Subprojects

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 subprojects, as the main project. See *Creating subprojects within projects* on page 3-57 for more information.

3.2 Overview of the project window

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


Metrowerks CodeWarrior
File Edit View Search Project Debug Tools Window Help
926_build_4.mcp
Release
Files
Link Order
Targets
dhrystone.mcp
initcm.s
init.s
Path: C:\Program Fil...\inittcm.s
Path: C:\Program Files\AR...\init.s
Line 1

File	Code	Data
init.s	0	0
vectors.s	0	0
embed_uish.s	0	0
inittcm.s	0	0
initmmu.s	0	0
initcache.s	0	0
dhry_1.c	0	0
dhry_2.c	0	0
retarget.c	0	0
13 files	0	0

File	Code	Data
dhry_1.c	0	0
dhry_2.c	0	0
2 files	0	0

```
Region_4K       EQU 2_0011
Region_8K       EQU 2_0100
Region_16K      EQU 2_0101
Region_32K      EQU 2_0110
Region_64K      EQU 2_0111
Region_128K     EQU 2_1000
Region_256K     EQU 2_1001
Region_512K     EQU 2_1010
Region_1M       EQU 2_1011

Region_Enable   EQU 2_1

                EXPORT InitTCMentry
```

```
Mode_UND          EQU     0x1B
Mode_SYS          EQU     0x1F ; available on ARM

I_Bit             EQU     0x80 ; when I bit is se
F_Bit             EQU     0x40 ; when F bit is se

Len_FIQ_Stack      EQU     256
Len_IRQ_Stack      EQU     256

Offset_FIQ_Stack          EQU     0
Offset_IRQ_Stack          EQU     Offset_FIQ_Sta
Offset_SVC_Stack          EQU     Offset_IRQ_Sta

; --- System memory locations

CM_ctl_reg        EQU     0x1000000C          ; A
Remap_bit         EQU     0x04                ; B
```

Figure 3-1 Typical CodeWarrior Project window

The project window uses three distinct views to display your files and build targets:

- the Files view
- the Link Order view
- the Targets view.

The following sections describe the project window in detail:

- *Navigating the project window*
- *Project views* on page 3-6.

3.2.1 Navigating the project window

To navigate the project window, use the vertical scroll bar on the right side of the window, or the Up and Down Arrow keys on your keyboard. If the project window contains many files, use the Home key to scroll to the top of the list, or use the End key to scroll to the end of the list.

Use the Page Up and Page Down keys to scroll one page up or one page down the project window. See *Selection by keyboard* on page 3-28 for information on how to select files as you type.

Using the Project Window toolbar

The toolbar in the project window has buttons and other items that provide shortcuts to commands and information about the project. You can choose the items to display on the toolbar, and the order in which those items are displayed. You can also choose to hide or display the toolbar itself. See *Customizing toolbars* on page 9-48 for more information on configuring toolbars in the CodeWarrior IDE.

3.2.2 Project views

The project window provides three distinct views on the files, groups, and subprojects that make up your project. These are:

- the Files view
- the Link Order view
- the Targets view.

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

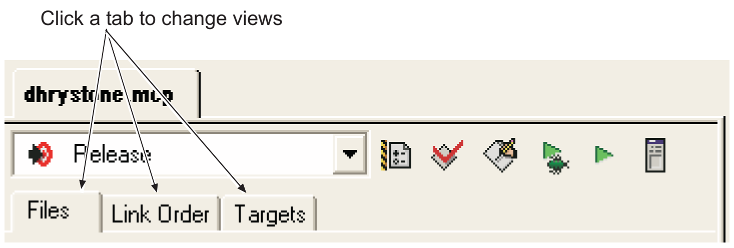


Figure 3-2 View tabs at the top of the project window

The project views are described in:

- *Files view*
- *Link Order view* on page 3-10
- *Targets view* on page 3-12.

Files view

The Files view (Figure 3-3 on page 3-7) shows a list of all the files, groups, and subprojects for all the build targets in the project. You can 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 information about modification status, file access paths, code size, data size, current build target, debugging status, and other information.

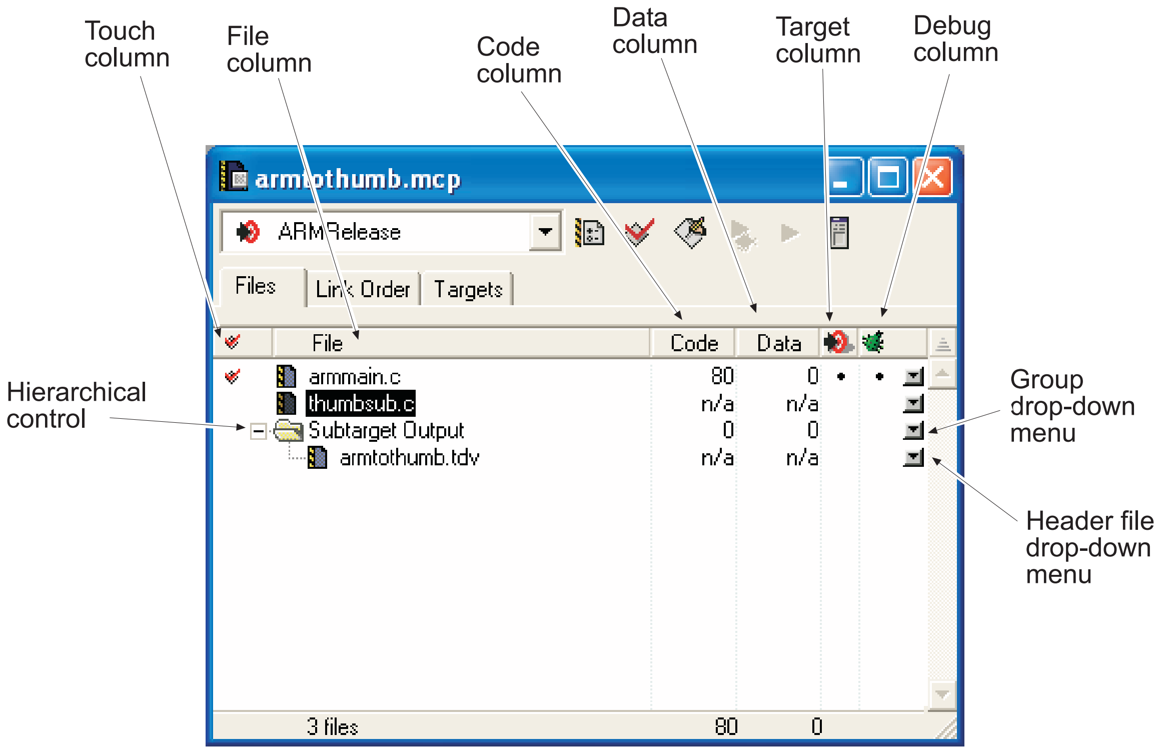


Figure 3-3 Project window Files view

The Files view window 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 panel.
- 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, it means that 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) of data, including 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, it means that 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.

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 ARM Debuggers* 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 3-49 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.

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 3-38 for more information.

Header Files drop-down menu

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

- lists and opens header files for your project source files
- 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. Select a file from the menu to open that file. 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 will differ depending on whether a filename or group name is selected.

Link Order view

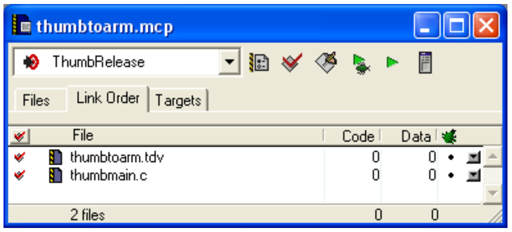
The Link Order view shows information about how the CodeWarrior IDE will compile and link the final output file for the current build target. Figure 3-4 shows an example.

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 in which 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 RealView linker with a list of object files in the order in which they are compiled. By default, the RealView linker processes object files in the order in which they are presented.

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-load description file. See *Configuring the RealView linker* on page 10-65 and the linker chapter of the *RealView Compilation Tools Linker and Utilities Guide* for more information.



File	Code	Data
thumbtoarm.tdv	0	0
thumbmain.c	0	0
2 files	0	0

Figure 3-4 Example Link Order view

The Link Order view tab displays columns that are 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 columns in the Link Order view are:

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 the description in *Files view* on page 3-6 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.

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 3-38 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 (Figure 3-5) shows information about the build targets in a project, and build target dependencies.

The Targets view shows a list of the build targets in the project. This view also shows the objects that the build targets depend on to create a final output file. Figure 3-5 shows an example Targets view with four targets which are defined by the ARM stationery. See *Working with multiple build targets and subprojects* on page 3-44 for a detailed description of the Targets view, and for information on working with build targets in general.

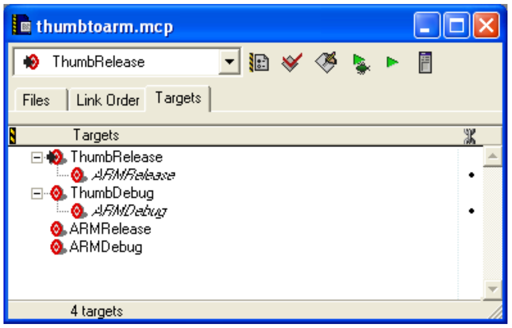


Figure 3-5 Example Targets view

3.3 Working with simple projects

This section describes the basic project operations:

- *Creating a new project*
- *Opening a project* on page 3-15
- *Closing a project* on page 3-17
- *Saving a project* on page 3-18
- *Choosing a default project* on page 3-19
- *Moving a project* on page 3-19
- *Importing and exporting a project as XML* on page 3-20.

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, though you should still review some configuration options to ensure that they are relevant to your development environment. This kind of project file is useful for quickly creating new projects.

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 3-27 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 (Figure 3-6 on page 3-14).

   ——— **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 used by the ARM version of the CodeWarrior IDE.

   ————————

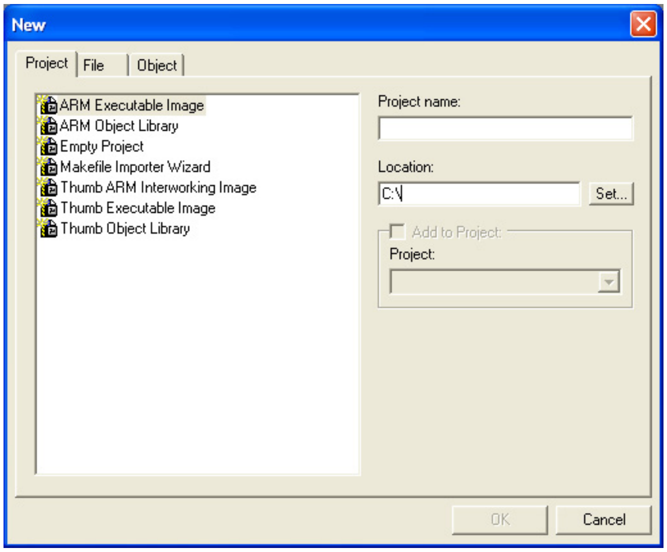


Figure 3-6 New dialog box

2. Ensure that the **Project** tab is selected and choose the project stationery file on which you want to base your own project. You can choose from:
   - 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 3-27 for more information on the project stationery files supplied by ARM. This is the recommended method if you are a new user.
   - The Makefile Importer Wizard. See *Converting GNU Makefiles* on page C-6 for information on using this wizard.

   ——— **Note** ———

   Using the keyboard

   You can use Ctrl+Tab or the left and right arrow keys to move between the **Project**, **File**, and **Object** tabs. To set the focus on the list of stationery:

   a. Press Ctrl+Tab until the Project tab is selected.

   b. Press Tab again.

   c. Press the Up and Down arrow keys to move to items in the stationery list.

   ———————————

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

4. Click **OK**. The CodeWarrior IDE creates a new project based on the project stationery you have selected. See:
   - *Adding files to a project* on page 3-29, for information on how to add your own source files to the new project
   - *Compiling and linking a project* on page 4-38 for more information on building your new project.

3.3.2 Opening a project

This section describes how to:

- open existing projects so you can work on them
- open subprojects from within a project window
- open 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 3-19 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 File dialog box (Figure 3-7).

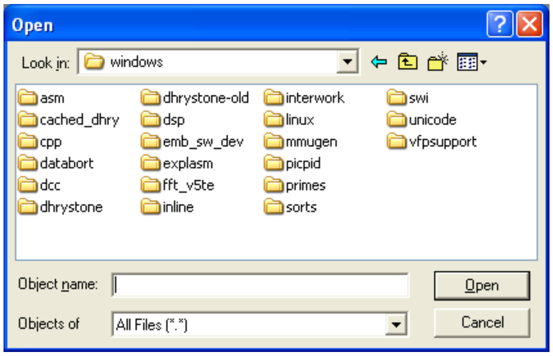


Figure 3-7 Open dialog

2. If not already set, use the **Files of Type** drop-down list 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.

   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 you have opened recently in the **File** menu. Use the **Open Recent** command in the **File** menu 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 subprojects from the project window

To open a subproject contained within your project, double-click the subproject file icon in the project window. The CodeWarrior IDE displays the subproject in a new project window. See *Working with multiple build targets and subprojects* on page 3-44 for more information on using subprojects.

Opening project files created on other host platforms

Project files are cross-platform compatible. For example, you can open and use a project created under Mac OS on a Windows machine.

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 file name 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` folder, from the other host platform to your computer.

3. Open the project in the CodeWarrior IDE and rebuild it.

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* on page 3-18 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.

Note

You can also save project window layout settings to a separate workspace file. For more information, refer to *Working with project workspaces* on page 3-25.

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 file name.

Caution

Do not attempt to make a copy of an open project from the Windows desktop. This can 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.

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 2-29 for more information.

3.4 Working with project windows

The CodeWarrior IDE provides you with options for how you wish to display the windows of a project. The options are:

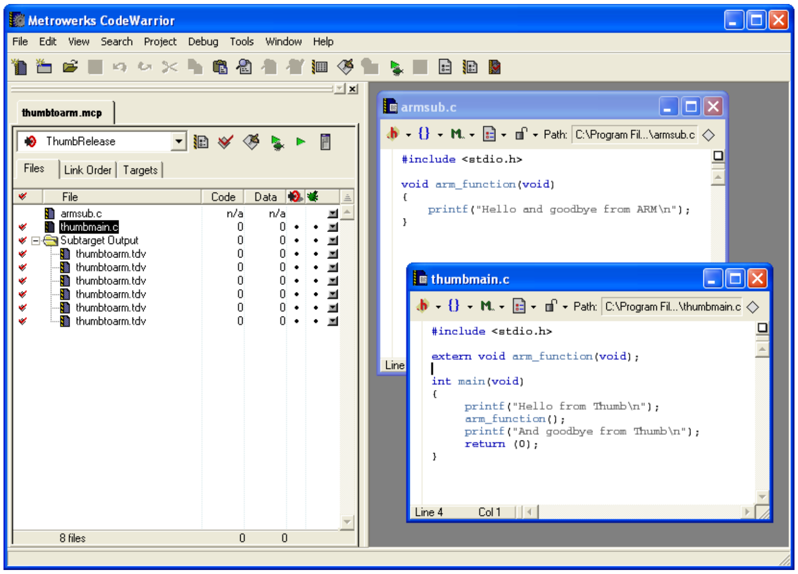
- *Multiple Document Interface (MDI)*; windows that are enclosed within the main IDE application window. See *Projects displayed using the MDI interface*.
- *Floating Document Interface (FDI)*; windows that can be moved over the entire Desktop. See *Projects displayed using the FDI interface* on page 3-23.

The MDI interface provides additional options to dock windows and also to float windows (similar to the FDI interface). The MDI interface is the default display option. See *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 3-25.

3.4.1 Projects displayed using the MDI interface

The MDI interface option displays all project windows enclosed within the main CodeWarrior IDE application window. See Figure 3-8:



✓	File	Code	Data		
	armsub.c	n/a	n/a		
✓	thumbmain.c	0	0	•	•
✓	Subtarget Output	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
✓	thumbtoarm.tdv	0	0	•	•
	8 files	0	0		

armsub.c

```
#include <stdio.h>

void arm_function(void)
{
    printf("Hello and goodbye from ARM\n");
}
```

thumbmain.c

```
#include <stdio.h>

extern void arm_function(void);

int main(void)
{
    printf("Hello from Thumb\n");
    arm_function();
    printf("And goodbye from Thumb\n");
    return (0);
}
```

Figure 3-8 Project displayed using the MDI interface

In addition, the MDI interface 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* an MDI window:

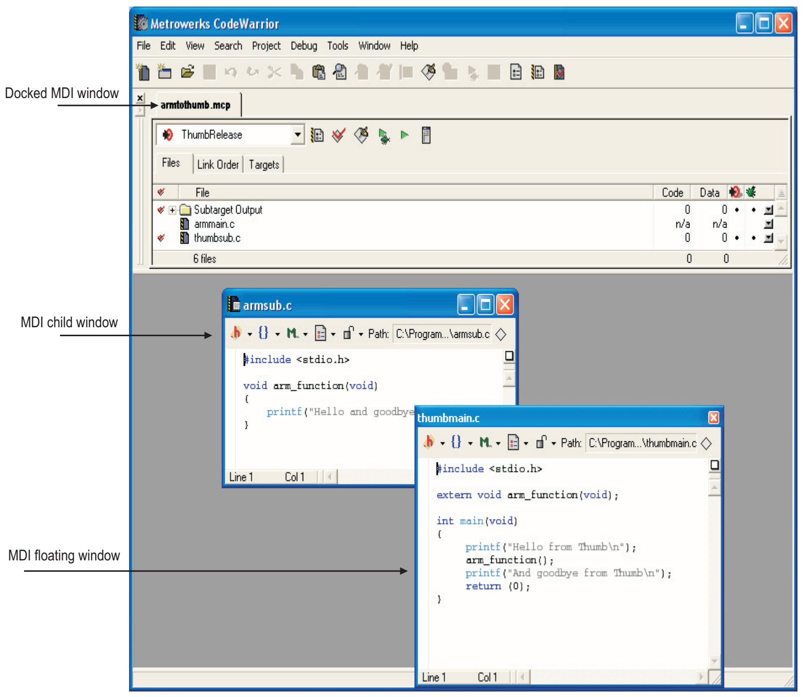
1. Right-click over the window title bar.

   A pop-up menu is displayed that provides the following choices:

   - **MDI child**
   - **Docked**
   - **Floating**

2. Choose **Docked** to dock the window, or **Float** to float the window.

Using the above procedure you can customize the display of MDI windows to suit your particular requirements. See Figure 3-9:



```
#include <stdio.h>

void arm_function(void)
{
    printf("Hello and goodbye
}
```

```
#include <stdio.h>

extern void arm_function(void);

int main(void)
{
    printf("Hello from Thumb\n");
    arm_function();
    printf("And goodbye from Thumb\n");
    return (0);
}
```

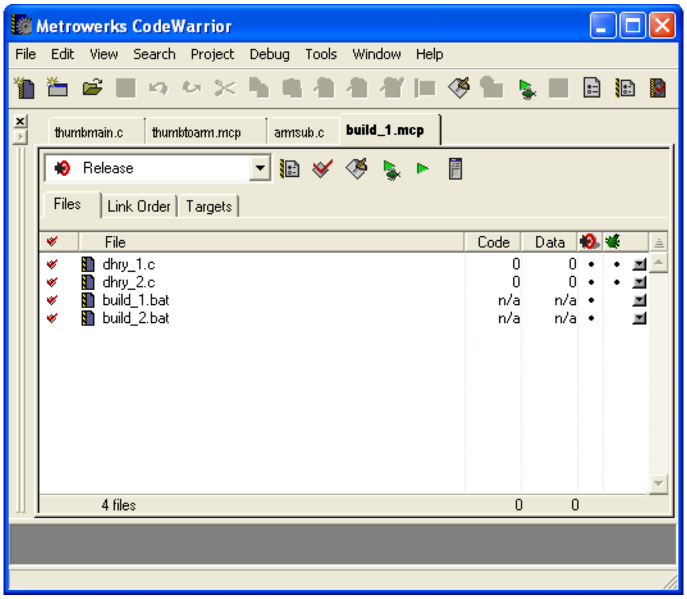
Figure 3-9 MDI interface showing all window display options

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

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

- right-click over the window and choose MDI child from the menu displayed.
- 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. For example:

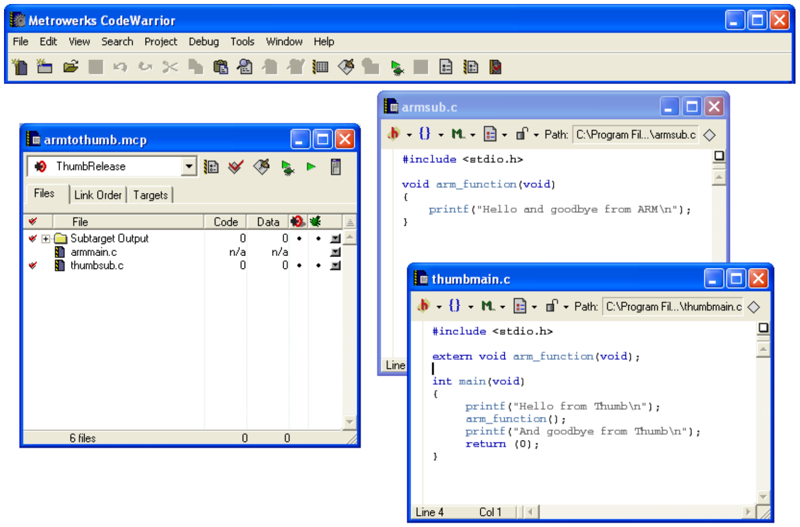


File	Code	Data
dhry_1.c	0	0
dhry_2.c	0	0
build_1.bat	n/a	n/a
build_2.bat	n/a	n/a
4 files	0	0

Figure 3-10 Docked and stacked file and project windows

3.4.2 Projects displayed using the FDI interface

The FDI interface option displays all open project windows as separate windows which can be moved around the desktop. The CodeWarrior IDE application window changes to show just the main menu bar and toolbar. Figure 3-11 on page 3-24:


Metrowerks CodeWarrior
File Edit View Search Project Debug Tools Window Help
armtothumb.mcp
ThumbRelease
Files
Link Order
Targets
armsub.c
Path: C:\Program Fil...\armsub.c
Line
thumbmain.c
Path: C:\Program Fil...\thumbmain.c
Line 4
Col 1

File	Code	Data
Subtarget Output	0	0
armmain.c	n/a	n/a
thumbsub.c	0	0
6 files	0	0

```
#include <stdio.h>

void arm_function(void)
{
    printf("Hello and goodbye from ARM\n");
}
```

```
#include <stdio.h>

extern void arm_function(void);

int main(void)
{
     printf("Hello from Thumb\n");
     arm_function();
     printf("And goodbye from Thumb\n");
     return (0);
}
```

Figure 3-11 Project displayed using the FDI interface

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

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 IDE can use a default workspace, or it can use a workspace that you create. The CodeWarrior works with one workspace at a time. You can save and re-apply a workspace from one 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, refer to *Configuring IDE extras* on page 9-9.

You can also create a workspace and apply this to replace of the current workspace. Saving your own workspace is useful if you have opened lots of project windows and wish to revert to your original project window 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 **Open** 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 may involve current projects being automatically closed and new projects being opened.

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 may involve current projects being automatically closed and new projects being opened.

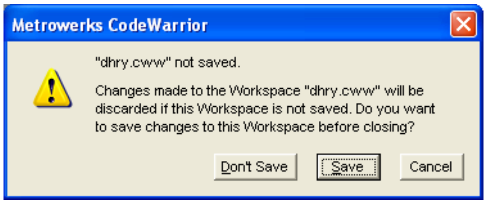
See *Configuring IDE extras* on page 9-9 for more information about configuring how many entries appear in the **Recent** menu.

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:



Metrowerks CodeWarrior

"dhry.cww" not saved.

Changes made to the Workspace "dhry.cww" will be discarded if this Workspace is not saved. Do you want to save changes to this Workspace before closing?

Figure 3-12 CodeWarrior 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.

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 folder for the project. It then copies all the files related to the stationery project to the new folder.

Project stationery can include:

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

3.6.1 The project stationery folder

The project stationery folder is located in:

`IDEs\CodeWarrior\CodeWarrior\5.6.1\1592\win_32-pentium\Stationery`.

where `1592` is the build number. (Assuming that the CodeWarrior IDE has been installed in the default installation directory of `c:\Program Files\ARM`.)

ARM-supplied project stationery files for common types of projects are located in subdirectories in the project stationery 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 and easily.

All the supplied stationery projects use:

- default target settings (little-endian ARM7TDMI)
- default *Procedure Call Standard for the ARM Architecture* PCS options for the compiler and assembler
- separate build targets for `Debug` and `Release` options (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*.

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 3-29
- *Grouping files in a project* on page 3-33
- *Moving files and groups* on page 3-35
- *Removing files and groups* on page 3-36
- *Touching and untouching files* on page 3-38.

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.

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 3-30 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 3-31 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 3-32.

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 3-33 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 dialog. See *Configuring file mappings* on page 10-24 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.

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 3-28 for more information on selecting files and groups of files.

Note

If you drag and drop a folder 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 3-35 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 an Add Files dialog box (Figure 3-13).

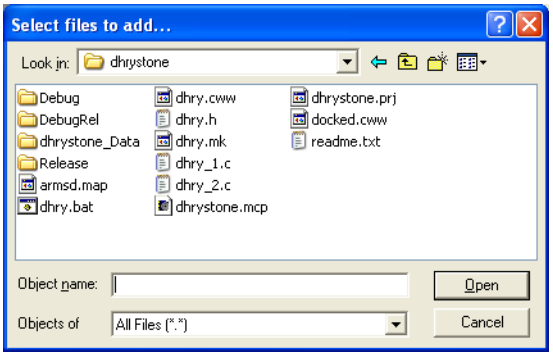


Figure 3-13 Adding files to a project

2. Use the **Files of type** drop-down list 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 target configuration panel. See *Configuring file mappings* on page 10-24 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 file name. Alternatively, double-click the file to add it to your project immediately.
   - To select multiple files, press the Control key and click on the file names in the dialog box.
   - To select a contiguous group of files, click on the first file name 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 to targets 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.

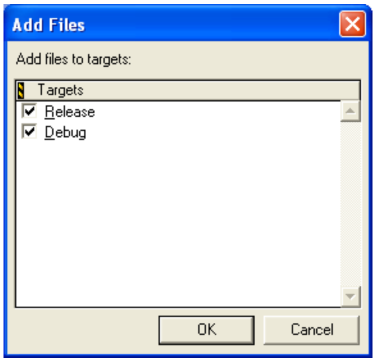


Figure 3-14 Add files to targets dialog box

Adding files with drag and drop

You can drag suitable files or folders 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 folder, the CodeWarrior IDE checks to ensure that the folder, or one of its subfolders, contains at least one file with a recognized filename extension, and that file is not already in the project. Folders 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 10-24 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 folders you want to add to the project.

   You can select files in many places, including the desktop or the multi-file 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.

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 to targets dialog box (see Figure 3-14 on page 3-31). 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 3-36 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.

——— **Note** ———

The **Add Window** menu item 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 file name extension, or is an unsaved window. See *Configuring file mappings* on page 10-24 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 to targets dialog box (see Figure 3-14 on page 3-31). 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 to a Windows folder. For example, if you drag a folder of source files onto the project window, the CodeWarrior IDE creates a new group with the same name as the folder. 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 (Figure 3-15 on page 3-34).

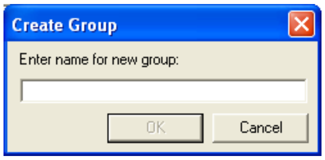


Figure 3-15 Create Group dialog

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).

   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.

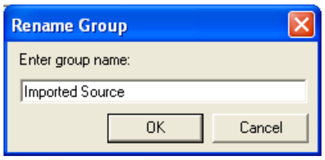


Figure 3-16 Rename group dialog

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.

- 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).

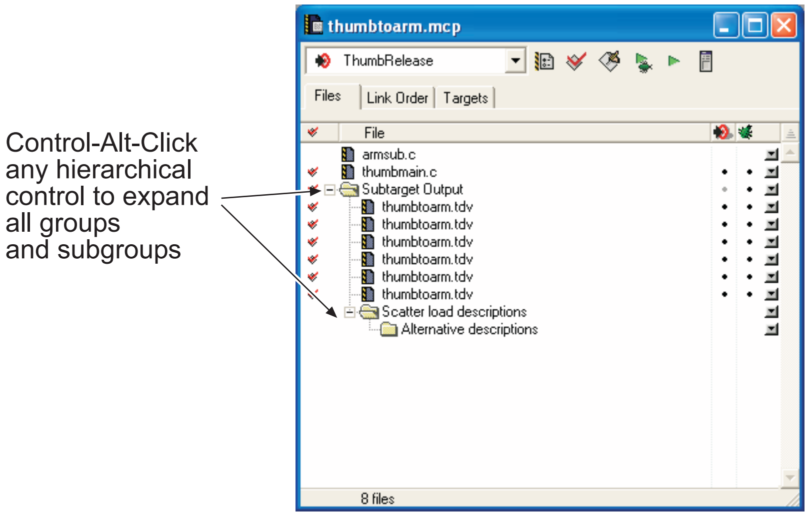


Figure 3-17 Expanding groups and subgroups

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 3-28 for more information.
2. Drag the selected files or groups to their new location in the project window. A focus bar (an underline) indicates where the selected files will be moved when the mouse button is released.
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 (Figure 3-18 on page 3-36).

Note

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. If the arrow appears to the right of the icon, the files are inserted into the target group.

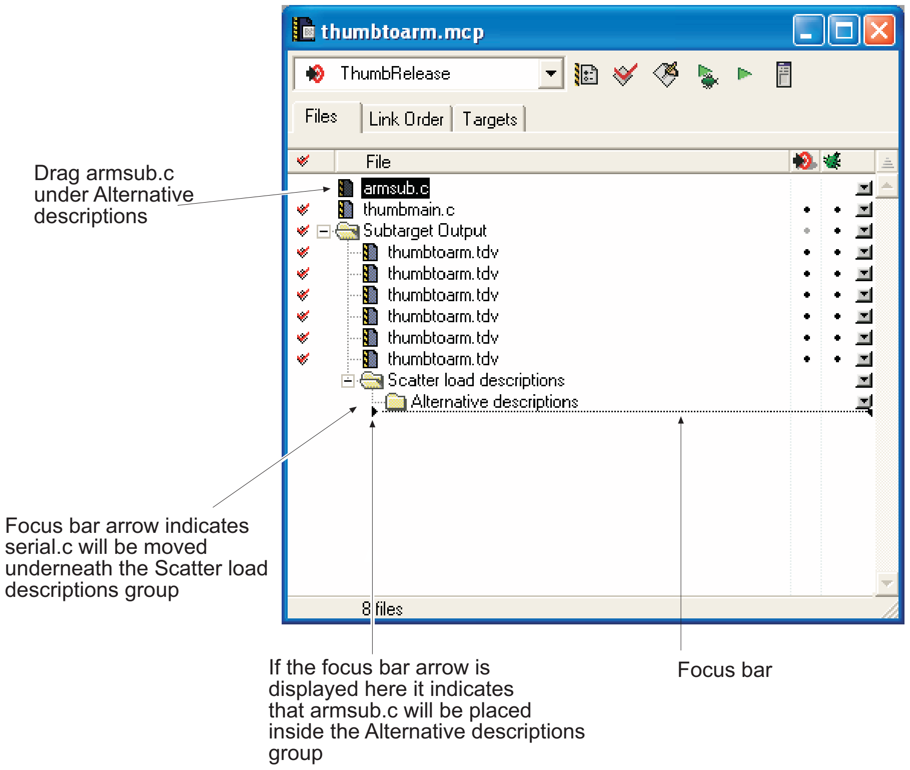

thumbtoarm.mcp
ThumbRelease
Files
Link Order
Targets
File
armsub.c
thumbmain.c
Subtarget Output
thumbtoarm.tdv
thumbtoarm.tdv
thumbtoarm.tdv
thumbtoarm.tdv
thumbtoarm.tdv
thumbtoarm.tdv
Scatter load descriptions
Alternative descriptions
8 files
Drag armsub.c under Alternative descriptions
Focus bar arrow indicates serial.c will be moved underneath the Scatter load descriptions group
If the focus bar arrow is displayed here it indicates that armsub.c will be placed inside the Alternative descriptions group
Focus bar

Figure 3-18 Moving a file

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.

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

1. Click either the **Files** view tab or the **Link Order** view 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.

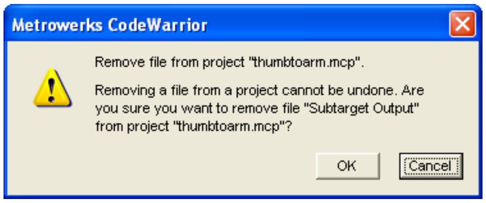
   Note

   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 3-28 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 (Figure 3-19).


Metrowerks CodeWarrior

Remove file from project "thumbtoarm.mcp".

Removing a file from a project cannot be undone. Are you sure you want to remove file "Subtarget Output" from project "thumbtoarm.mcp"?

Figure 3-19 Remove file confirmation dialog

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 3-29.

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 3-7).

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 been 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 3-20 on page 3-39).

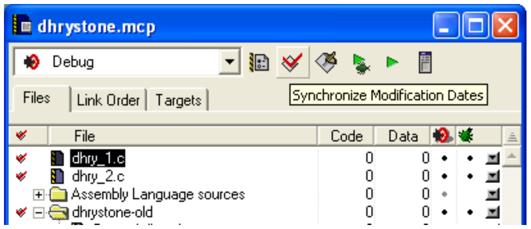


Figure 3-20 Synchronize modification dates

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.

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:

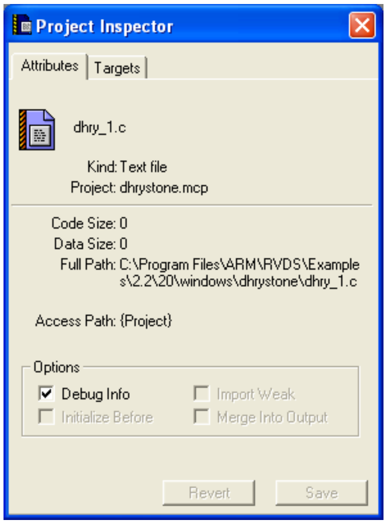
- An Attributes panel that displays file attributes such as the name and location of the file, the code and data size of the file, if it has been compiled, and whether or not debug information is generated for the file.
- A Targets panel that displays a list of the build targets that include a specific 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 3-21 on page 3-40).

   You can use this panel to specify whether debug information is generated for the current file when it is compiled. See *Generating debug information* on page 5-4 for more information on configuring debug information for files.

   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.


Project Inspector

Attributes | Targets

dhry_1.c

Kind: Text file
Project: dhrystone.mcp

Code Size: 0
Data Size: 0
Full Path: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\dhrystone\dhry_1.c

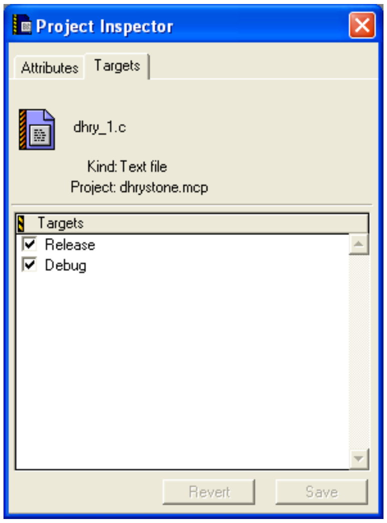
Access Path: {Project}

Options
- [✓] Debug Info
- [ ] Import Weak
- [ ] Initialize Before
- [ ] Merge Into Output

Figure 3-21 Project Inspector window for attributes

3. Click the **Targets** tab to display the Targets panel (see Figure 3-22) and select the build targets that you want to include the current file.

   See *Assigning files to build targets* on page 3-49 for more information on how to select files for inclusion in specific build targets.


Project Inspector

Attributes | Targets

dhry_1.c

Kind: Text file
Project: dhrystone.mcp

Targets
- [✓] Release
- [✓] Debug

Figure 3-22 Project Inspector window for targets

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

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 source control system is in use
- projects that use embedded subprojects 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 dialog.

  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 subproject 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 subdirectory of the directory in which the CodeWarrior IDE is installed is added recursively to your system paths.

  See *Setting access path options* on page 10-13 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, 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. 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 10-13 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 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 10-20 for more information on configuring cache modification dates option.

3.9 Working with multiple build targets and subprojects

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

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

- release code
- debug code

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 3-48
- *Assigning files to build targets* on page 3-49
- *Changing a build target name* on page 3-51
- *Assigning build target dependencies* on page 3-52
- *Building all targets in a project* on page 3-56
- *Creating subprojects within projects* on page 3-57.

3.9.1 Overview of complex projects

Complex projects are projects that contain either multiple build targets, or multiple subprojects, or both. You can use multiple build targets and subprojects 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 3-52 for detailed information on defining dependencies between multiple build targets.

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

You can link the output from any build target in a subproject 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 subprojects within projects* on page 3-57 for detailed information on using subprojects.

Strategy for creating complex projects

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

Project structure

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

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 subproject. For example, you can include a subproject 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 subproject 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 subproject is an externally built library which is built using a different build environment but is still required as part of the main project.

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. Subprojects 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 multi-file 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 subproject unless the subproject 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 3-56 for more information.

  ———————————

  See *Compiling and linking a project* on page 4-38 for more information on building your project.

To set the active build target in a project:

From the CodeWarrior IDE menu bar

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

From the Build Target drop-down list

Select the build target from the **Build Target** drop-down list (Figure 3-23).

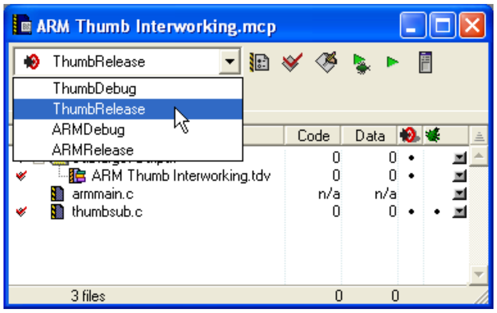


Figure 3-23 Build target drop-down list

From the Targets View

Click once on the name of a build target to select it as the current build target. Current build targets that the CodeWarrior IDE will build are denoted by a circular icon (an archery target) with an arrow going into it (Figure 3-24).

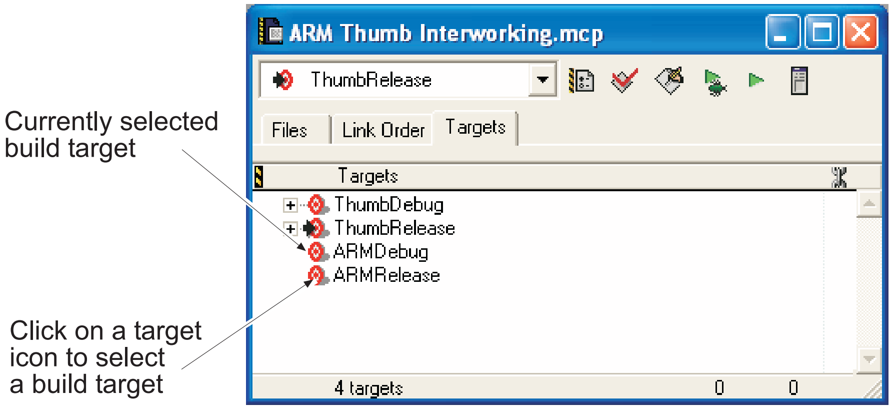
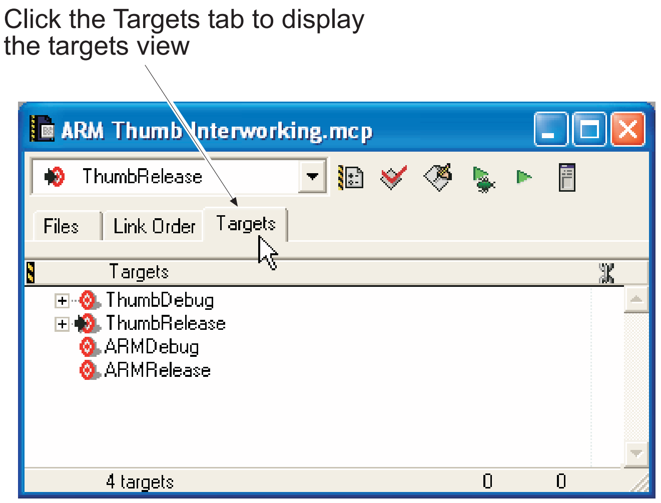

Currently selected build target
Click on a target icon to select a build target
ARM Thumb Interworking.mcp
ThumbRelease
Files
Link Order
Targets
Targets
ThumbDebug
ThumbRelease
ARMDebug
ARMRelease
4 targets

Figure 3-24 Targets view

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 3-25).


Click the Targets tab to display the targets view
ARM Thumb Interworking.mcp
ThumbRelease
Files
Link Order
Targets

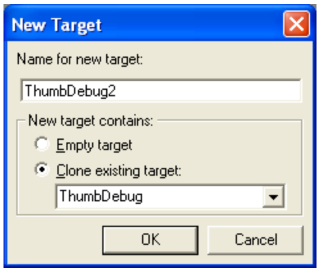
Targets
- ThumbDebug
- ThumbRelease
- ARMDebug
- ARMRelease

4 targets 0 0

Figure 3-25 Targets view tab

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

   The CodeWarrior IDE displays a New Target dialog box (Figure 3-26).



Name for new target: ThumbDebug2

New target contains:
Empty target
[✓ Clone existing target:] ThumbDebug

Figure 3-26 New Target dialog box

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

4. Select the type of new 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 build 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 3-52 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.

You can associate the new build target with other build targets to create dependent build relationships.See *Assigning build target dependencies* on page 3-52 for more information on defining build target dependencies.

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
- the Project inspector.

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

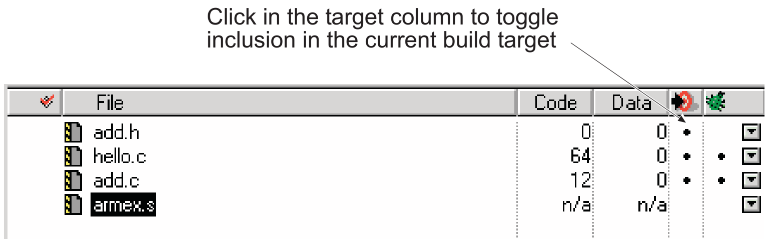
Including a file in a build target using the Target column

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 3-27 on page 3-50).

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 3-46 for information on how to change the active build target.

2. Click in the **Target** column next to the file (Figure 3-27):
   - 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.


Click in the target column to toggle inclusion in the current build target

	File	Code	Data			
	add.h	0	0	•		
	hello.c	64	0	•	•	
	add.c	12	0	•	•	
	armex.s	n/a	n/a			

Figure 3-27 The target column

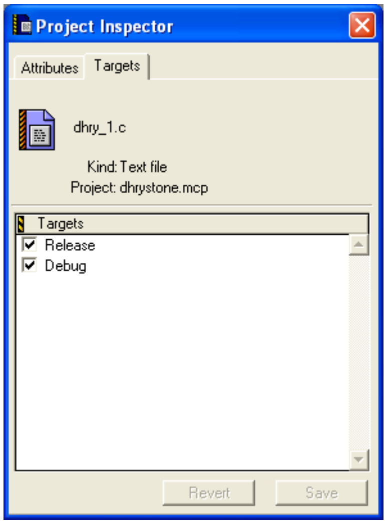
Note

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 in step 1 (Figure 3-24 on page 3-47).



dhry_1.c

Kind: Text file
Project: dhrystone.mcp

Targets
- [x] Release
- [x] Debug

Figure 3-28 Project Inspector window for targets

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

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

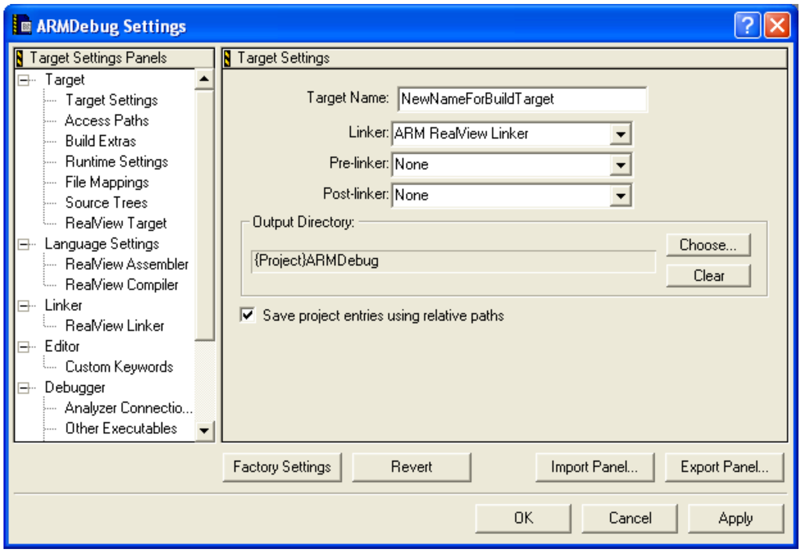
3.9.4 Changing a build target name

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 3-25 on page 3-48).

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 (see Figure 10-4 on page 10-9).

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 3-29 on page 3-52).



Target Settings

Target Name: NewNameForBuildTarget

Linker: ARM RealView Linker

Pre-linker: None

Post-linker: None

Output Directory: {Project}ARMDebug

✓ Save project entries using relative paths

Figure 3-29 Renaming a build target

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

You can also open the Target Settings window by selecting **Target 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.

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

- *Creating a new build target* on page 3-48 for more information on creating build targets.
- *Setting the current build target* on page 3-46 for more information on setting the current build target.
- *Strategy for creating complex projects* on page 3-45 for information on strategies for setting up complex projects with build targets and subprojects.

To create a new, dependent build 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 3-48.)

   The new Target is displayed in the list of targets (Figure 3-30). By default, the new target is not dependent on any existing targets

   Figure 3-30 shows an example.

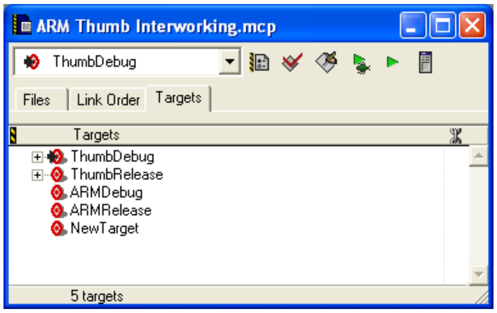


Figure 3-30 Creating a new build target

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 (Figure 3-31).

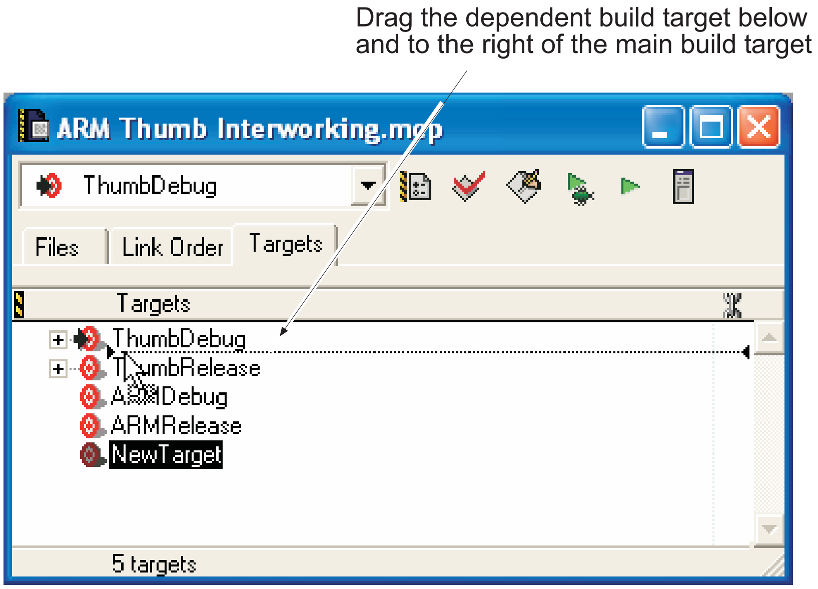

Drag the dependent build target below and to the right of the main build target
ARM Thumb Interworking.mcp
ThumbDebug
Files
Link Order
Targets
Targets
ThumbDebug
ThumbRelease
ARMDebug
ARMRelease
NewTarget
5 targets

Figure 3-31 Dragging a 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 3-32). 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. See the following steps for more information.

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

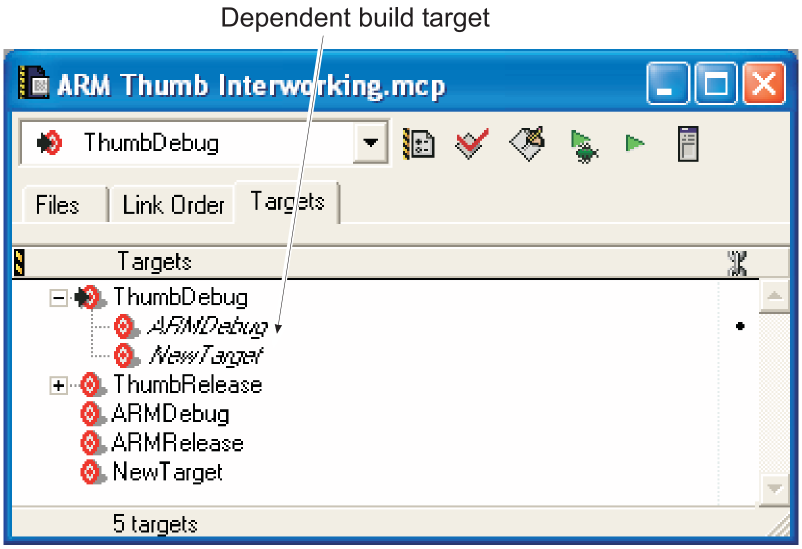
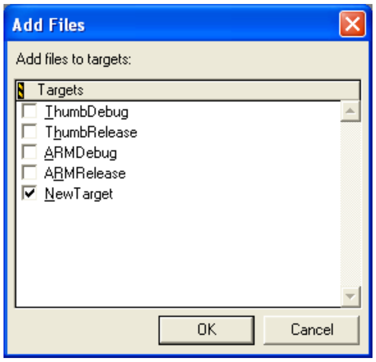


Figure 3-32 Dependent build target

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 (Figure 3-33 on page 3-55).


Add Files

Add files to targets:

Targets
☐ ThumbDebug
☐ ThumbRelease
☐ ARMDebug
☐ ARMRelease
☑ NewTarget

Figure 3-33 Adding files to the 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 (Figure 3-34 on page 3-56). This is most commonly done when the output of the subtarget is a library, partially linked object, or directory of objects.

   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.

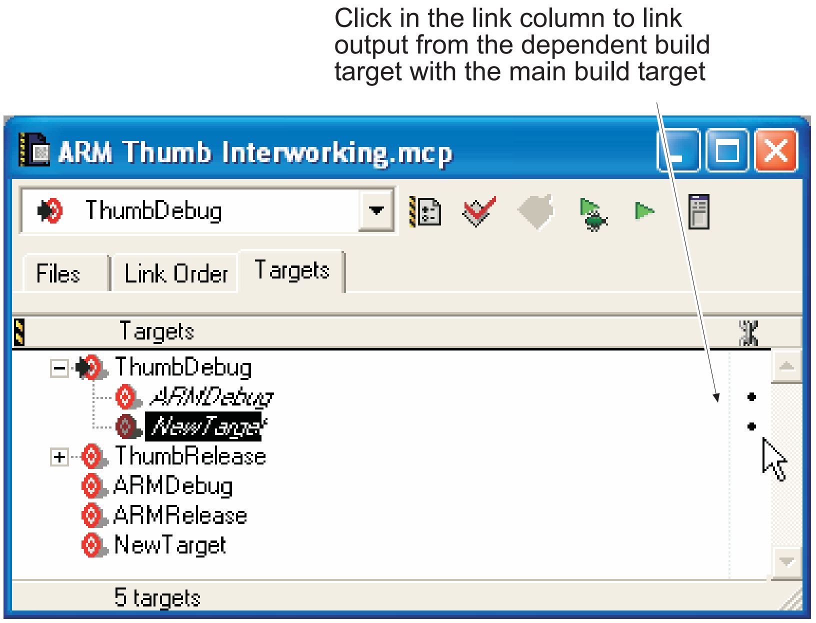

Click in the link column to link output from the dependent build target with the main build target
ARM Thumb Interworking.mcp
ThumbDebug
Files
Link Order
Targets
Targets
ThumbDebug
ARMDebug
NewTarget
ThumbRelease
ARMDebug
ARMRelease
NewTarget
5 targets

Figure 3-34 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 6.

3.9.6 Building all targets in a project

The CodeWarrior IDE does not have 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.

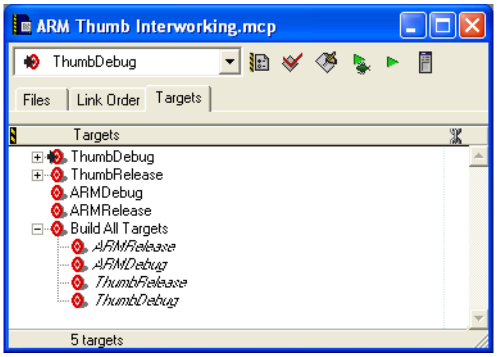
To create a Build All Targets build target:

1. Create a new build target using the Empty Target option in the new target dialog. See *Creating a new build target* on page 3-48 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.

3. Drag the existing build targets underneath and to the right of the new build target listing in Targets view to create a build dependency. See *Assigning build target dependencies* on page 3-52 for more information on creating dependent build targets.

The Build All Targets build target displays an italicized list of dependent build targets. Figure 3-35 shows an example based on the ARM stationery. The new build target is named Build All Targets.



ThumbDebug
ThumbRelease
ARMDebug
ARMRelease
Build All Targets
ARMRelease
ARMDebug
ThumbRelease
ThumbDebug

Figure 3-35 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 3-46 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 subprojects within projects

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

Subprojects are listed in the Files view of the project window with the other components of your project. They can be assigned to any build target in the main project. When you add a subproject, you can select the build targets to which it belongs.

You can configure your main project so that a **Make** command builds one or more build targets in a subproject when it builds the containing 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 subproject to any build target in the main project. This means, for example, that you can link the `Release` build target output from a subproject 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 subproject* on page 3-58
- *Subprojects with multiple build targets* on page 3-58

For an example of creating a subproject refer to *Creating a project with a subproject* on page 4-26.

3.9.8 Main steps required for compiling and linking a subproject

The following steps are required to compile and link the output from a subproject with your main project:

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

Each of these steps is independent of the others. For example, you can specify that:

- a subproject build target is built when the main project is built, but not linked with the main project
- the output from a subproject build target is linked, but the subproject build target is not built when the main project is built.

3.9.9 Subprojects with multiple build targets

If you have multiple build targets in a subproject, you can create link dependencies between the main project and any of the build targets in that subproject (see Figure 3-36 on page 3-59).

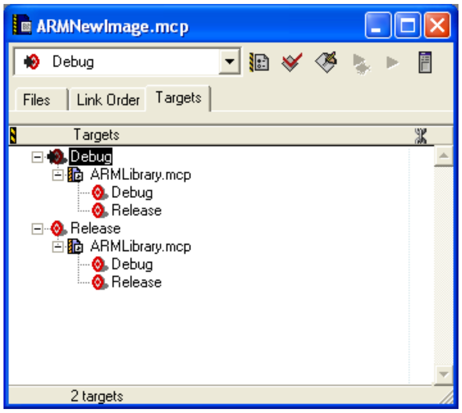


Figure 3-36 Multiple build targets in subprojects

The subprojects 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 3-27), you can create a link dependency to any, or all, of the `Debug` and `Release` build targets in the subproject, all of which might contain the same code, but built with different optimization and debug options.

When you build your project, the RealView 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.

Refer to *Creating a project with a subproject* on page 4-26 for an example of creating a project within an existing project.

You should now be completely familiar with the basics of CodeWarrior projects. Refer to the next chapter for more advanced information about projects and in particular information about the use of the ARM project stationery.

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 the previous introductory chapter about how projects are used in the CodeWarrior IDE.

This chapter contains the following sections:

4.1 About the ARM Project stationery

The following stationery projects are 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 3-13. 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. This stationery project is configured to use:

- The compiler to compile all C or C++ files in ARM state. If you want to compile C code in C++ mode, you must set the Source Language to **ISO Standard C++ (1998)** in the **Source** tab of the RealView Compiler configuration panel (see *Configuring the RealView compiler* on page 10-47 for more information).
- The assembler to assemble all files with a `.s` filename extension.
- The linker to link a simple executable ELF image.
- The RealView Debugger to both 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. This stationery project is configured to use:

- The compiler to compile all Thumb C or C++ files in Thumb state. If you want to compile Thumb C code in Thumb C++ mode, you must set the Source Language to **ISO Standard C++ (1998)** in the **Source** tab of the RealView Compiler configuration panel (see *Configuring the RealView compiler* on page 10-47 for more information.
- The 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 linker to link a simple executable ELF image.
- The RealView Debugger to both debug and run executable images output by the project.

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:

- it 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 Toolkit Utilities chapter of 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 will contain ELF object format members. This stationery is similar to the Thumb executable image stationery. The major differences are:

- it is configured to use the ARM RealView Librarian (`armar`) utility to output an object library.
- you cannot debug or run a standalone library file until it is linked into an image.

See the Toolkit Utilities chapter of 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 project to build an executable ELF image from interworking ARM and Thumb code. This stationery project 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 3-49 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.
- 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.

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. The build targets are `ThumbRelease`, `ThumbDebug` and `ARMRelease` and `ARMDebug`.

The `ARMRelease` build target is a dependency of `ThumbRelease`, and the `ARMDebug` build target is a dependency of `ThumbDebug`.

The default build target for Thumb ARM interworking stationery is `ThumbDebug`.

See *Working with multiple build targets and subprojects* on page 3-44 for more information on using the ARM project stationery to create complex projects. See *Configuring debug and optimization* on page 10-55 for information on how to set debug and optimization options.

4.2 Working with an existing ARM project

This section describes how to work with an existing CodeWarrior 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* on page 4-6
- *Running the image in the Debugger* on page 4-6.

4.2.1 Opening an existing CodeWarrior project

To open an existing CodeWarrior 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 project 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, CodeWarrior creates directories for them in the project directory. The output directory is the folder where CodeWarrior places the final output file it builds a target in a project. This example has two build targets, so two build output directories will be created called `Debug` and `Release`.

4.2.2 Building the image for your project

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

When you build an image for a particular build target, CodeWarrior for RVDS 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`

4.2.3 Running the image in the Debugger

To run the image for your project:

1. Select the debugger to use for the project (see *Selecting a debugger for the project*).
2. Prepare the connection between your debugger and the target (see *Preparing the debugger connection* on page 4-7).
3. Run the project image (see *Running the project image* on page 4-8)

Selecting a debugger for the project

You can choose the debugger that your project is to use by default. This can be one of the following:

- RealView Debugger (this is the default for new projects, and for the example RVDS projects)
- AXD
- `armsd`.

Note

AXD and `armsd` are supported for legacy ARM7 and ARM9 targets only.

You can set up the required debugger on the following panels (see *Configuring the debugger* on page 10-83 for details):

- **RealView Debugger**
- **RealView Runner**.

To select a debugger for your project:

1. Select **Edit** → ***TargetName*** **Settings...** from the main menu to display the settings panels for the current build target, where ***TargetName*** is the current build target.
2. Select **RealView Debugger** in the Debugger panels of the Target Settings Panels list.
3. In the **Choose Debugger** tab, select the debugger you want to use. The default is **RVD (RealView Debugger)**.
4. Select **RealView Runner** in the Debugger panels of the Target Settings Panels list.
5. In the **Choose Debugger** tab, select the debugger you want to use. The default is **RVD (RealView Debugger)**.

Note

For RealView Debugger, there is no difference between starting it from the **RealView Debugger** panel or from the **RealView Runner** panel.

Preparing the debugger connection

Before you can run your image in the debugger, you must set up a connection between the debugger and the required target. How you do this depends on the debugger you are using, see:

- *Preparing RealView Debugger*
- *Preparing the ARM eXtended Debugger* on page 4-8
- *Preparing the ARM Symbolic Debugger* on page 4-8.

Preparing RealView Debugger

To prepare a connection in RealView Debugger:

1. Start RealView Debugger:

   Start → Programs → ARM → RealView Developer Suite v2.2 → RealView Debugger v1.8

2. Set up the connection to your debug target.

   For details on setting up a connection to a debug target in RealView Debugger, see *RealView Debugger Essentials Guide* and the *RealView Debugger Target Configuration Guide*.

3. Exit RealView Debugger.

When you next start RealView Debugger from CodeWarrior, the connection is established and your image is loaded automatically.

Preparing the ARM eXtended Debugger

To prepare a connection in AXD, for example:

1. Start AXD:

   Start → Programs → ARM → RealView Developer Suite v2.2 → AXD Debugger v1.3.1

2. Set up the connection to your debug target. The default is ARM7TDMI.

   For details on setting up a connection to a debug target in AXD, see *RealView Developer Suite AXD and armsd Debuggers Guide*.

3. Exit AXD.

When you next start AXD from CodeWarrior, the connection is established and your image is loaded automatically.

Preparing the ARM Symbolic Debugger

To prepare a connection for `armsd`, do the following from the **Choose Debugger** tab (see *Selecting a debugger for the project* on page 4-6):

1. Select **armsd (ARM Symbolic Debugger)**.
2. Select the **Armsd1** tab.
3. Select the required Debug Target.
4. Select the required Target Processor.
5. Click **OK** to save the settings and close the settings panel.

Running the project image

To run the image in the chosen debugger:

1. Prepare the debugger connection if you have not done this already (see *Preparing the debugger connection* on page 4-7).
2. Run the debugger using one of the following options:
   - Select **Project → Run** from the main menu, to run the debugger you set up in the **RealView Runner** panel.
   - Select **Project → Debug** from the main menu, to run the debugger you set up in the **RealView Debugger** panel.

The debugger starts, establishes the connection you have set up, and loads the project image to the target. Run and debug the image using the features provided with the debugger. For details on how to use the debugging features for the chosen debugger, see:

- the RealView Debugger documentation set
- the *RealView Developer Suite AXD and armsd Debuggers Guide*.

4.3 Using the Thumb ARM interworking stationery

The Thumb ARM interworking 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 subprojects* on page 3-44 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. The project also has ARM build targets as dependencies of Thumb build targets.

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

1. Preparing the tutorial (see *Before you begin*)
2. Create the project (see *Creating an ARM Thumb interworking project* on page 4-11)
3. Adding your source files to the build targets (see *Adding source files to the build targets* on page 4-13)
4. Building the image (see *Building the image* on page 4-15)
5. Running the image in the debugger (see *Running the image in the debugger* on page 4-15)

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
```

Sources used in this 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 project:

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

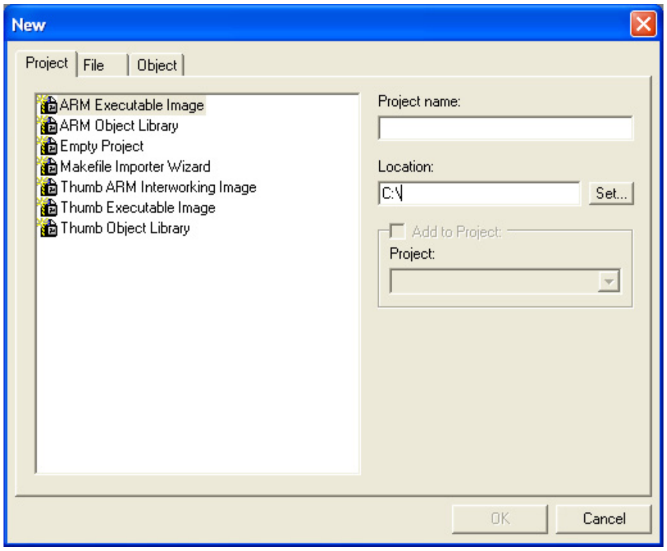
- Use the following source files for an image with `main()` in Thumb state
  - `thumbmain.c`
  - `armsub.c`

Do not copy these sources to the `Tutorial` directory at the moment. You are instructed to do this as part of the tutorial.

4.3.2 Creating an ARM Thumb interworking project

To create the project for ARM Thumb interworking:

1. Select **File** → **New** from the main menu to display the New dialog box, shown in Figure 4-1.



ARM Executable Image
ARM Object Library
Empty Project
Makefile Importer Wizard
Thumb ARM Interworking Image
Thumb Executable Image
Thumb Object Library

Project name:

Location: C:\

Figure 4-1 New project dialog box

2. Select the **Thumb ARM Interworking Image** stationery. See *About the ARM Project stationery* on page 4-2 for information about the stationery supplied with RVDS.

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. If you are using the `Tutorial` directory you created in *Before you begin* on page 4-10, select:

   `C:\Myprojects\Tutorial`

4. In the Create New Project dialog box:
   a. Enter the project name **ARM Thumb Interworking** in the File name field.
   b. Make sure that the Save as type is set to **Project Files (*.mcp)**.
   c. Make sure that the **Create Folder** checkbox is selected.
   d. Click **Save** to close the dialog box.
5. The Project name and Location fields in the New project dialog are set to the values you specified in the Create New Project dialog box.

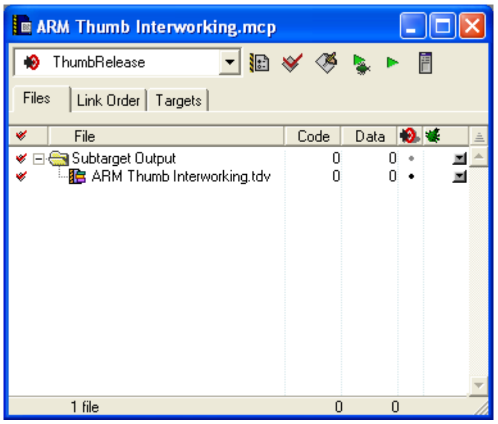
   Note

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

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

CodeWarrior for RVDS:

- Creates a project directory in the location you specified in step 3. This directory has the same name as the project name you specified in step 4a.
- Sets up the project with the settings defined in the Thumb ARM Interworking Image stationery template.
- Displays the Files view for the new project, shown in Figure 4-2.



File	Code	Data
Subtarget Output	0	0
ARM Thumb Interworking.tdv	0	0
1 file	0	0

Figure 4-2 Interworking project Files view

Note

A file with the extension `.tdv` is created as the output for each subtarget. This is an intermediate build file for that subtarget, which CodeWarrior 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 sources for the interworking project, you must:

1. Copy the source files to the project directory (see *Copying the source files to the project directory*).
2. Add the ARM source files to the ARM build targets (see *Adding the ARM sources*).
3. Add the Thumb source files to the Thumb build targets (see *Adding the Thumb sources* on page 4-14).

——— **Note** ———

When you open a project, if any messages are produced, then the Project Messages dialog box has focus. To be able to add files to your project, you must first select the project tab (see Figure 4-2 on page 4-12 for an example).

———————————

For more information about all the methods available for adding files to a project, see *Adding files to a project* on page 3-29.

Copying the source files to the project directory

Copy the `armmain.c` and `thumbsub.c` source files from the RVDS `interwork` example directory (see *Sources used in this tutorial* on page 4-10) to the project directory `*directory*\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. CodeWarrior for RVDS displays an **Add Files** dialog.
2. Deselect the Thumb build targets and click **OK** (Figure 4-3 on page 4-14).

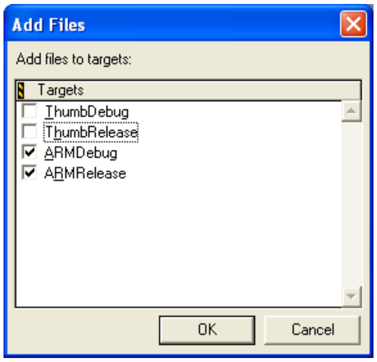
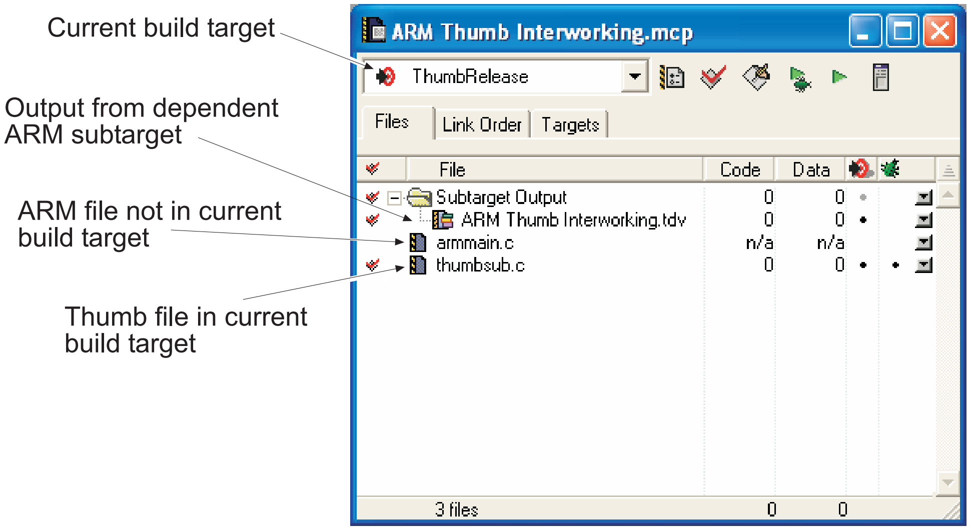


Figure 4-3 Add ARM files

CodeWarrior for RVDS adds the file to the ARM build targets only.

Adding the Thumb sources

Add your `thumbsub.c` source file to the Thumb build targets. Follow the same procedure as that described in *Adding the ARM sources* on page 4-13, 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 in the project. Files not in the current build target do not have a black dot in the Target column, and have a code size of `n/a`.


Current build target
Output from dependent ARM subtarget
ARM file not in current build target
Thumb file in current build target
ARM Thumb Interworking.mcp
ThumbRelease
Files
Link Order
Targets

File	Code	Data
Subtarget Output	0	0
ARM Thumb Interworking.tdv	0	0
armmain.c	n/a	n/a
thumbsub.c	0	0
3 files	0	0

Figure 4-4 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.

CodeWarrior for RVDS:

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 **Targets** drop-down list. If you select an ARM build target, only the ARM part of your project source is built.

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

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

You can change this location as described in *Changing the build location*.

Changing the build location

If you want to change the build location:

1. Select **Edit** → ***TargetName*** **Settings...** from the main menu to display the *`TargetName`* Settings panel.
2. In the Target Settings 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
   ```

4.3.5 Running the image in the debugger

To run the image in your chosen debugger, follow the instructions given in *Running the image in the Debugger* on page 4-6.

4.4 Creating binary ROM images

The default project stationery supplied by ARM Limited 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 panel to call `fromelf` as a post-linker. This is the default setting.
2. Configure the **RealView fromELF** 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 panel to create the image structure you require, or use a scatter-load 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*.

Also, see the tutorial *Creating a project that builds an object library and ROM images* on page 4-17.

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

This tutorial demonstrates how to create a project that:

- builds an object library
- builds an executable image from the object library and other source files
- converts the executable image to ROM images.

This section contains:

- *Summary of the project*
- *Creating the CodeWarrior project* on page 4-18
- *Setting up the build target for the library* on page 4-19
- *Creating the application build target* on page 4-21
- *Assign the build target dependencies* on page 4-22
- *Adding the sources to the appropriate build targets* on page 4-23
- *Creating the application.bin ROM image* on page 4-24
- *Creating the application.m32 ROM image* on page 4-25.

4.5.1 Summary of the project

This tutorial uses the `*install_directory*\ARM\RVDS\Examples\...\vfpsupport\Build4` example project of RVDS. Analysis of the build options in `build.bat` or `makefile` shows that two build targets are required to build the entire project. The build targets are listed in Table 4-1.

Table 4-1 Required build targets and their purpose

Build target	Purpose
`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 4-25.
`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 subsequently referred to as a subtarget.

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 project

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:

   `directory``\vfpsupport`

   If you want to, you can remove the `build1`, `build2`, `build3`, `build5`, `build6`, `build7`, and `util` subdirectories from your copy.

   Note

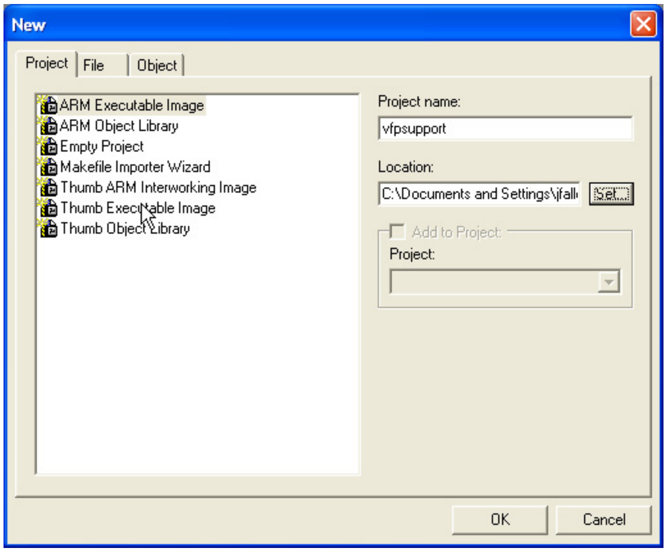
   See (see the *RealView Developer Suite Getting Started Guide*) for more information about the example projects included with RVDS.

2. Start CodeWarrior for RVDS, and close any other projects that are open.
3. Select **File** → **New...** from the main menu.
4. Select **ARM Executable Image**.
5. Click **Set...** to display the Create New Project dialog box.
6. Locate your *`directory`*`\vfpsupport` folder.

   Note

   If you want to use this project to create stationery, you must create the project file in your *`directory`*`\vfpsupport` folder, rather then the `build4` subdirectory (see *Creating your own project stationery* on page 4-32 for more details). You might also want to remove any files that are not required, such as `*.bat` and `makefile` from your backup directory.

7. Enter **vfpsupport**, and deselect **Create Folder**. See Figure 4-5 on page 4-19.



Project name: vfpsupport

Location: C:\Documents and Settings\jfall

Project:

Figure 4-5 New project for tutorial

8. Click **Save**.

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 is to use a build target to build the `vfp_support.a` library. To set up the build target for the library:

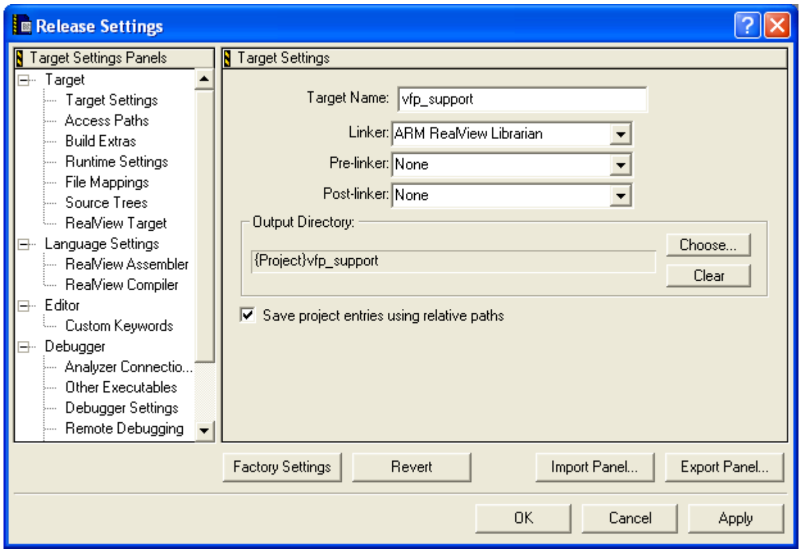
1. Select the **Targets** tab.

2. Double-click on the `Release` build target to display the Release Settings dialog box.

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`.

f. Click **Apply** to apply the settings.

The build target name changes to `vfp_support`.



Target Name: vfp_support

Linker: ARM RealView Librarian

Pre-linker: None

Post-linker: None

Output Directory: {Project}vfp_support

☑ Save project entries using relative paths

Figure 4-6 Release Settings panel

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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Enter **--fpmode ieee_full** in the `Equivalent Command Line` after any other command line options.
   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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Click the **Debug/Opt** tab.

   d. Set the Optimization Level to **2 (poor debug view, better code)**.
   e. Set the Floating Point Models to **Full IEEE**.
   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 list.

4.5.4 Creating the application build target

To create the `application` build target:

1. Double-click on the `Debug` target to display the Debug Settings dialog box.
2. Select **Target Settings** in the Target Settings Panels list:
   a. Change the Target Name to **application**.
   b. Set Linker to **ARM RealView Linker**.
   c. Make sure Post-linker is set to **ARM 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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Enter **--fpmode ieee_full** in the `Equivalent Command Line` after any other command line options.
   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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Click the **Debug/Opt** tab.

   d. Set the Optimization Level to **2 (poor debug view, better code)**.
   e. Set the Floating Point Models to **Full IEEE**.
   f. Select the **Preprocessor** tab.
   g. Enter **USE_SERIAL_PORT**, then click **Add**.
   h. 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** for 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 **Plain binary** is the Output format. This is the default setting.
   b. Enter **application.bin** as the Output file name.
   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 Assign the build target dependencies

To assign the build targets that are to be subtargets 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 subtarget is created under `application`.

3. Expand the `application` build target.

4. For the `vfp_support` subtarget, click the link column so that a black dot appears opposite the subtarget.

5. Set the `application` build target to be the current build target. See Figure 4-7 on page 4-23

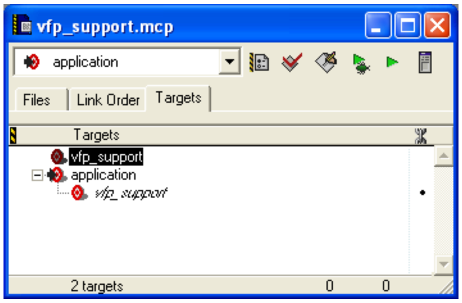


Figure 4-7 Assigning build target dependencies

4.5.6 Adding the sources to the appropriate build targets

Table 4-2 shows the source files that must be added to each build target.

Table 4-2 Source files belonging to each build target

Build target	Source files	Location
`vfp_support`	`*.c` `*.s`	*directory*`\vfpsupport\vfp_support`
`application`	`main.c`	*directory*`\vfpsupport`
	`*.c` `*.s`	*directory*`\vfpsupport\build4`
	`*.c` `*.s`	*directory*`\vfpsupport\vfp_init`

To add the source files to the appropriate build target, do the following:

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.

   Note

   To select multiple files, press the Ctrl key and select the required files.

3. Click **Open**.
4. Select the build target to which you want to add the file. Make sure you deselect all the other build targets.
5. Click **OK**.

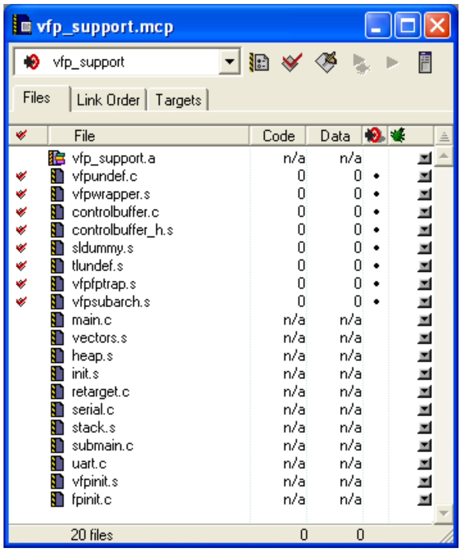
6. Repeat steps 1 to 5 for each directory listed in Table 4-2 on page 4-23.

Confirming the files are added to the correct build targets

To confirm that you have added the files to the correct 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 4-2 on page 4-23):
   - 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 that build target. The remaining source files are built when the associated subtargets are built. See Figure 4-8.
4. Repeat step 2 and 3 for the remaining build targets.



✓	File	Code	Data	
	vfp_support.a	n/a	n/a	
✓	vfpundef.c	0	0	•
✓	vfpwrapper.s	0	0	•
✓	controlbuffer.c	0	0	•
✓	controlbuffer_h.s	0	0	•
✓	sldummy.s	0	0	•
✓	tlundef.s	0	0	•
✓	vfpfptrap.s	0	0	•
✓	vfpsubarch.s	0	0	•
	main.c	n/a	n/a	
	vectors.s	n/a	n/a	
	heap.s	n/a	n/a	
	init.s	n/a	n/a	
	retarget.c	n/a	n/a	
	serial.c	n/a	n/a	
	stack.s	n/a	n/a	
	submain.c	n/a	n/a	
	uart.c	n/a	n/a	
	vfpinit.s	n/a	n/a	
	fpinit.c	n/a	n/a	
	20 files	0	0	

Figure 4-8 File view for current build target

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 the `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 application Settings dialog box.
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 file name.
   c. Click **Apply** to apply the settings.
   d. Click **OK** when CodeWarrior informs you 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.

4.6 Creating a project with a subproject

The following example shows how to modify the project created in *Creating a project that builds an object library and ROM images* on page 4-17. 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 4-27).
3. Add the sources to the build target for the new library project (see *Adding the sources to the build target* on page 4-28).
4. Remove the `vfp_support` build target, and associated source files (see Table 4-2 on page 4-23), from the `vfpsupport` project (see *Removing the vfp_support build target from the main project* on page 4-28).
5. Add the library project as a subproject to the `vfpsupport` ARM executable image project (see *Adding a project as a subproject* on page 4-29).

The example also gives details on how to specify link and build dependencies.

4.6.1 Creating the library project

To create the library project:

1. Start CodeWarrior for RVDS, and close any other projects that are open.
2. Select **File** → **New** from the main menu.
3. Select the **ARM Object Library** stationery.
4. Click **Set...** to display the Create New Project dialog box.
5. Locate your *`directory`*`\vfpsupport\vfp_support` folder.
6. Enter **vfp_support**, and deselect **Create Folder**.
7. Click **Save**.
8. Click **OK**. The `vfp_support.mcp` project file and the `vfp_support_Data` directory are created.

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 Release Settings dialog box.
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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Enter **--fpmode ieee_full** in the Equivalent Command Line after any other command line options.
   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 such as an **ARM1020E** or an **ARM1176JZF-S**.
   b. Set the Floating Point to **VFPv2**.
   c. Click the **Debug/Opt** tab.
   d. Set the Optimization Level to **2 (poor debug view, better code)**.
   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.

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 `*directory*\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, CodeWarrior automatically adds the sources 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:

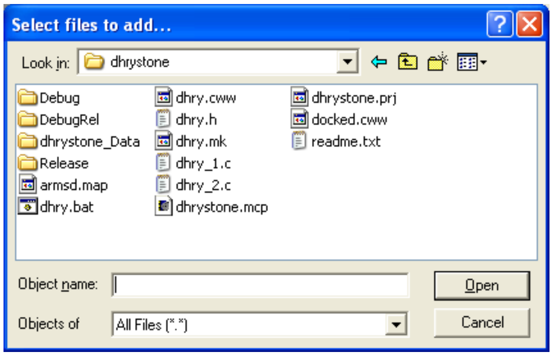
1. Close the new `vfp_support` library project.
2. Open the main `vfpsupport` project.
3. Click the **Files** tab to display the Files view for the project.
4. Change the current build target to be `vfp_support`. Those files that do not have `n/a` in the Code and Data columns are part of the `vfp_support` build target.
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. Click the **Targets** tab to display the Targets view for the project.
8. Right-click on the `vfp_support` subtarget, and select **Delete** from the context menu to delete it.
9. Click **OK** when prompted.

   The build target is removed from the project. There is now only one target in the project.

4.6.5 Adding a project as a subproject

To add the `vfp_support` library project as a subproject to the `vfpsupport` project:

1. Add the subproject 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.



Select files to add...

Look in: dhrystone

Debug, DebugRel, dhrystone_Data, Release, armsd.map, dhry.bat, dhry.cww, dhry.h, dhry.mk, dhry_1.c, dhry_2.c, dhrystone.mcp, dhrystone.prj, docked.cww, readme.txt

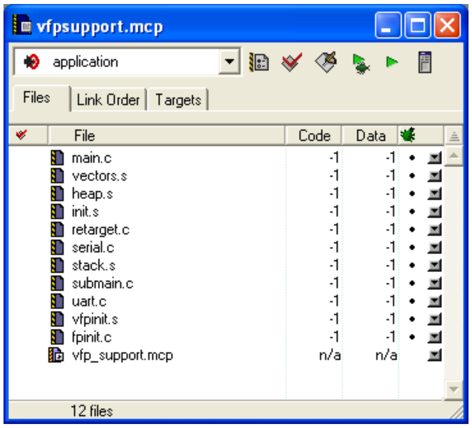
Object name:

Objects of: All Files (*.*)

Figure 4-9 Adding a subproject

   b. Locate the *`directory`*`\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 subproject to the `application` build target. Figure 4-10 shows an example of the Files view for the project.



vfpsupport.mcp

application

File	Code	Data
main.c	-1	-1
vectors.s	-1	-1
heap.s	-1	-1
init.s	-1	-1
retarget.c	-1	-1
serial.c	-1	-1
stack.s	-1	-1
submain.c	-1	-1
uart.c	-1	-1
vfpinit.s	-1	-1
fpinit.c	-1	-1
vfp_support.mcp	n/a	n/a

12 files

Figure 4-10 Project with subproject Files view

2. Click the **Targets** tab to display the Targets view for the project and click the plus sign next to a build target containing the subproject to expand the hierarchy. Each build target in the subproject is listed in the hierarchy. Figure 4-11 shows an example.

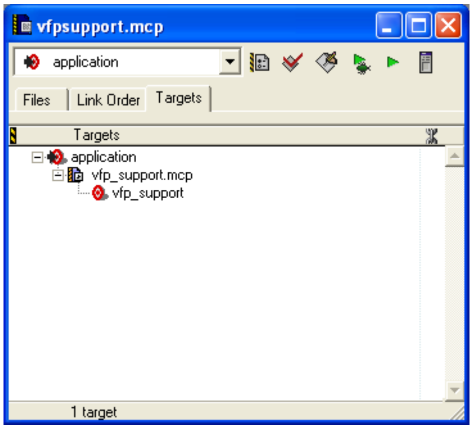


Figure 4-11 Subproject build target view

3. Click on the Target icon next to the subproject build target. This ensures that this subtarget is built when the main project is built (Figure 4-12 on page 4-31). CodeWarrior for RVDS displays an arrow and target icon for build targets that are selected. When the main project is built, selected subtargets are built first if they have changed, or have been touched.

   For example, in Figure 4-12 on page 4-31 the `vfp_support` subtarget in the example subproject is built before the `application` build target in the main project.

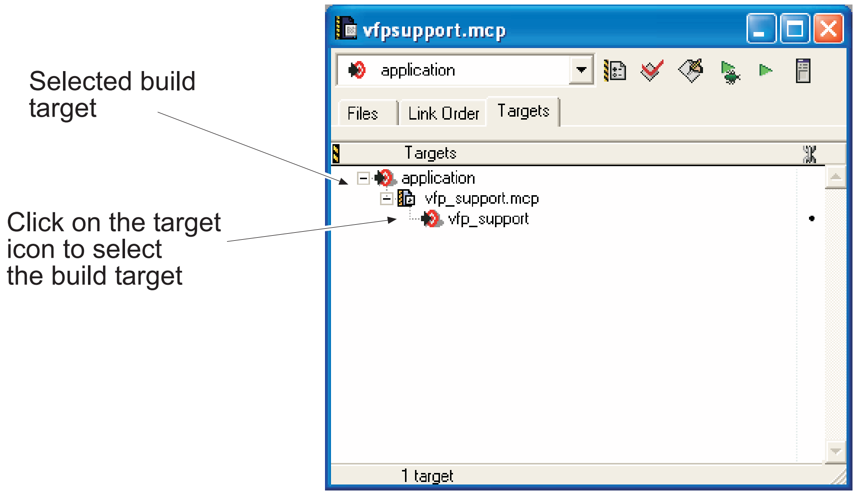


Figure 4-12 Selecting subproject build targets for building

4. You can now rebuild the project.

4.7 Creating your own project stationery

You can create your own CodeWarrior for RVDS stationery project file that includes:

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

You might want to do this if, for example, you have multiple developers working on different parts of a project, and you want to have them use common configuration settings.

The easiest way to create custom stationery is to modify a project based on existing stationery and save it under a new name in the stationery folder. CodeWarrior for RVDS 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 folder. Instead, create a new project in a user defined folder and then save it under a new name in the stationery folder.

4.7.1 Rules for creating project stationery

When creating project stationery, follow these rules:

- You must create a new folder for your project stationery, and give it a suitable name.
- The project file for your new stationery must be placed in the top-level folder of that stationery.
- 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 folder.
- You do not have to copy or duplicate any `<projectname>_Data` directories in the stationery folder.

4.7.2 Creating project stationery

This example uses the `vfpsupport` project described in *Creating a project that builds an object library and ROM images* on page 4-17.

To create your own custom stationery:

1. Open an existing project, or create a new project (see *Creating an ARM Thumb interworking project* on page 4-11), for example, `vfpsupport.mcp`.
2. Modify the project settings to suit your requirements. You can add and remove files as necessary to create the base project you want.
3. Select **Save A Copy As…** from the **File** menu. CodeWarrior for RVDS displays a Save a copy of project as dialog box.
4. Locate the CodeWarrior `Stationery` folder (see *The project stationery folder* on page 3-27):
5. Create a new folder for your stationery:
   a. Click the **Create New Folder** button on the dialog box.
   b. Enter a name for your new stationery folder, for example, **VFP Support**.
   c. Double-click on the new folder to make it the current folder. The folder name appears in the Save in field.
6. 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 folder.
7. Click **Save** to save the file.
8. Copy the project source files to the project stationery folder 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 4-32).

   ————————

   For the `vfpsupport` project:

   a. Copy the following directories from the `vfpsupport` directory to the new `VFP Support` stationery folder:
      - `build4`
      - `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 panel of the build targets.

---

b. Copy the `main.c` file from the `vfpsupport` directory to the new `VFP Support` stationery folder.

c. Remove the following files from the `VFP Support\build4` stationery folder:
   - `build.bat`
   - `makefile`.

d. Remove the following files from the `VFP Support\vfp_support` stationery folder:
   - `build.bat`
   - `makefile`
   - `vfp_support.a`.

Your new project stationery is now ready to use. You can select your custom project stationery 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.

4.8 Converting ARM projects to Thumb projects

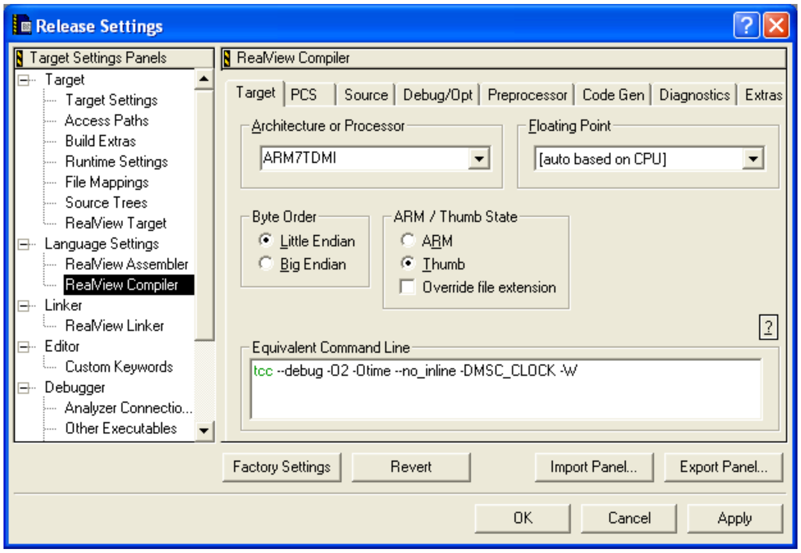
To convert an existing ARM project to Thumb:

——— **Note** ———

You must change the following configuration options for each build target in the project.

———————————

1. Open the project you want to convert.
2. Select ***TargetName*** **Settings…** from the **Edit** menu and click **RealView Compiler** in the Target Settings Panels list. The CodeWarrior IDE displays the RealView Compiler panel (Figure 4-13).
   a. Click the **Thumb** option in the **ARM / Thumb State** group box. See Figure 4-13:



Architecture or Processor: ARM7TDMI

Floating Point: [auto based on CPU]

Byte Order: [✓ Little Endian] Big Endian

ARM / Thumb State: ARM [✓ Thumb]

Override file extension: ☐

Equivalent Command Line:

```
tcc --debug -O2 -Otime --no_inline -DMSC_CLOCK -W
```

Figure 4-13 RealView compiler panel

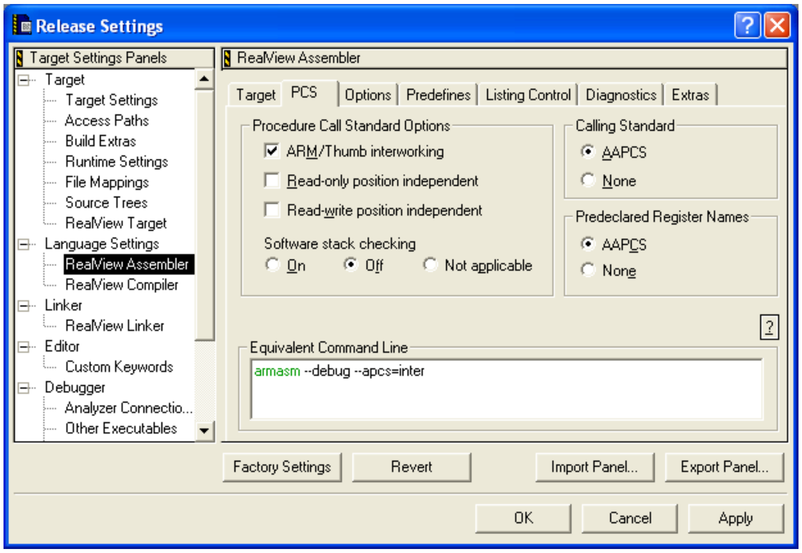
   b. Click the **PCS** tab and select **ARM/Thumb interworking** in the **Procedure Call Standard Options** group box.

      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. Click **Apply** to apply the changes.

3. Click **RealView Assembler** in the Target Settings Panel list to display the Language Settings panel for the RealView Assembler:

   a. Click the **PCS** tab and select **ARM / Thumb Interworking** in the **Procedure Call Standard Options** group box (Figure 4-14).



Procedure Call Standard Options
- [✓] ARM/Thumb interworking
- [ ] Read-only position independent
- [ ] Read-write position independent

Software stack checking: On [✓ Off] Not applicable

Calling Standard: [✓ AAPCS] None

Predeclared Register Names: [✓ AAPCS] None

Equivalent Command Line

```
armasm --debug --apcs=inter
```

Figure 4-14 Select Thumb initial state

   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.

   b. Ensure that the other language settings for the assembler are appropriate for your project.

   c. Click **Apply** to apply your changes.

4. Repeat all the above steps for each build target in the project.

5. Rebuild your project. The CodeWarrior IDE uses the compiler in Thumb state to rebuild your code.

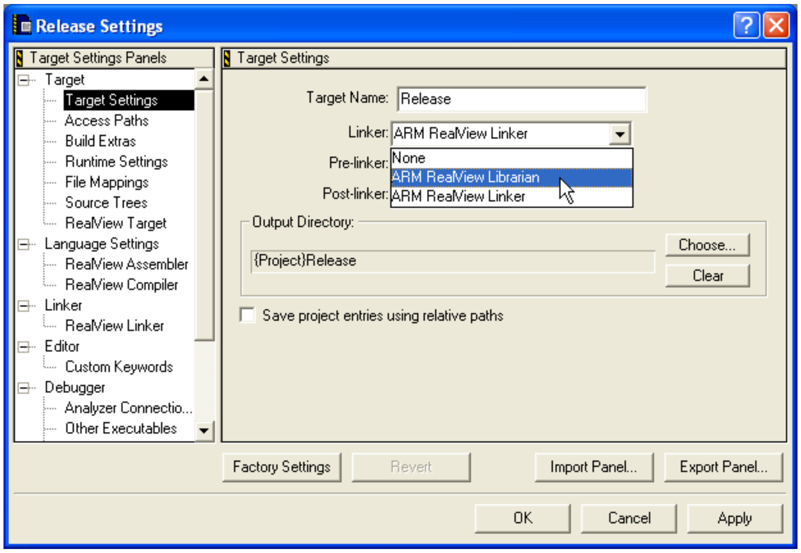
4.9 Converting Executable Image projects to Library projects

To convert an Executable Image project to a Library project you must change the following configuration options *for each build target in the project*:

1. Open the project you want to convert.

2. Select ***TargetName* Settings…** from the **Edit** menu and click **Target Settings** in the Target Settings Panels list. The CodeWarrior IDE displays the Target Settings panel

3. Click the **Linker** drop-down list and select **ARM RealView Librarian** (Figure 4-15).



Release Settings

Target Settings

Target Name: Release

Linker: ARM RealView Linker

Pre-linker:

Post-linker:

Output Directory: {Project}Release

Save project entries using relative paths

Figure 4-15 Select the ARM RealView librarian

4. Click **Apply** to apply your changes.

5. Repeat all the above steps for each build target in your project.

6. Rebuild your project. The CodeWarrior IDE calls the ARM RealView Librarian to create an object library.

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 compiling or linking 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 will use. See *Choosing a default project* on page 3-19 for more information.

This section describes:

- *Overview of compiling and linking*
- *Compiling files* on page 4-40
- *Making a project* on page 4-44
- *Removing objects from a project* on page 4-45.

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 lists. See *Overview of the project window* on page 3-4 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 will switch 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.

To modify this behavior you can edit the **ARM / Thumb State** group box in the **Target** tab in the **Target Settings** panel for the RealView Compiler. For example, you can force the compiler to compile all C and C++ sources using the RealView Compiler in ARM state, making no adjustments for any Thumb filename extensions. See *Configuring the RealView compiler* on page 10-47 for more information.

To force all C and C++ sources to be compiled as C only or C++ only, set the **Source Language** option in the **Source** tab in the **Target Settings** panel for the RealView Compiler. The options available are:

- ISO Standard C (1990)
- ISO Standard C++ (1998)

See *Configuring the RealView compiler* on page 10-47 for more information.

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 3-46 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 3-52 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 panel. See *Configuring target settings* on page 10-8 for more information. Table 4-3 describes the output file naming conventions and default locations.

Table 4-3 Default output names and locations

Output	Naming convention	Default location in the project folder
Executable ELF image	`*Project Name*.axf`	`Target_Name`
Partially linked ELF object	`*Project Name*.o`	`Target_Name`
ARM library	`*Project Name*.a`	`Target_Name`
Object code	`*filename*.o`	`Project_Name_Data\Target_Name\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. You can use the CodeWarrior IDE to compile:

- the current editor window
- one or more selected files in a project
- all the files in a project.

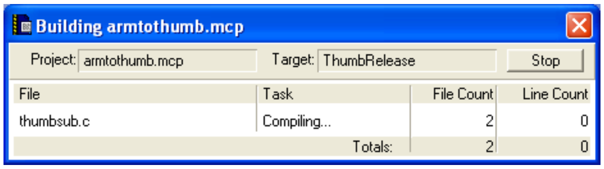
The object files generated by the compilation are placed in a data subdirectory of your main project folder. See *Making a project* on page 4-44 for information on compiling and linking your source files and libraries.

Note

The CodeWarrior IDE compiles the files only for the currently selected build target. See *Setting the current build target* on page 3-46 for more information on setting the build target for a compilation.

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 (Figure 4-16 on page 4-41). The Build Progress window displays the name of the file currently being compiled.



Project: armtothumb.mcp
Target: ThumbRelease

File	Task	File Count	Line Count
thumbsub.c	Compiling...	2	0
	Totals:	2	0

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 3-28 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.

   ————————————

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 3-46 for more information on setting the build target for a compilation.

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 3-38 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 3-38 for more information.

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 a **Errors & Warnings** window. (See *Using the message window* on page 5-11 for more information.)

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:

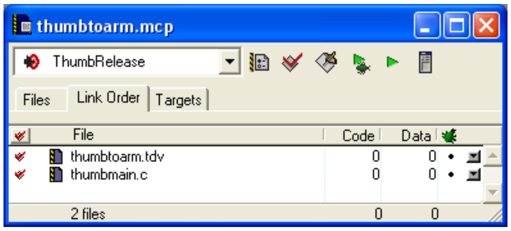
- compiling newly added, modified, and touched source files to produce ELF object files
- linking object files and libraries to produce an ELF image file, or a partially linked object, or library (using the ARM RealView Librarian)
- performing any postlink operations that you have defined for your build 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 postlink 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. The Link Order view is displayed in the project window (Figure 4-17).



File	Code	Data
thumbtoarm.tdv	0	0
thumbmain.c	0	0
2 files	0	0

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 3-10 for more information.

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 10-65 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 10-65 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 folder. The **Remove Object Code** command 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. Use the CodeWarrior object code removal facility described below.

Removing object code

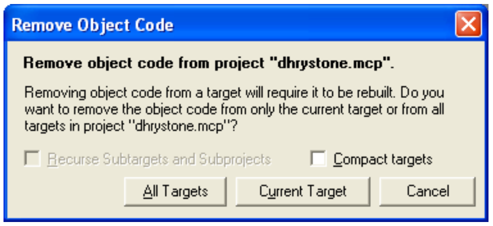
To remove the object code from a project:

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 will remove object code from the current default project, regardless of whether the file displayed in the current editor window belongs to that project.

2. Select **Remove Object Code** from the **Project** menu. The CodeWarrior IDE displays the Remove Object Code dialog box (Figure 4-18).



Remove object code from project "dhrystone.mcp".

Removing object code from a target will require it to be rebuilt. Do you want to remove the object code from only the current target or from all targets in project "dhrystone.mcp"?

Figure 4-18 Remove Objects dialog box

3. Select the object code removal options:
   - **Recurse Subtargets and Subprojects**, to remove object code data from all subtargets and subprojects that belong to the parent project. (If there are no subtargets or subprojects this option is greyed out.)
   - **Compact targets**, to also remove extraneous project data such as:
     - — Target data files (`.tdt` files)
     - — Browser data
     - — Dependency information

4. Click either:
   - **All Targets**, to remove all object code data for all build targets in the project, resetting the Code and Data size of each file in the project window to zero. (If there is only one build target this option is greyed out.)
   - **Current Target**, to remove the objects for the current build target only. See *Setting the current build target* on page 3-46 for more information on changing the build target.
   - **Cancel**, to cancel the operation so that object code is not removed.

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 4-48
- *Creating libraries with the ARM RealView Librarian* on page 4-49

4.11.1 Disassembling code

You can configure the CodeWarrior IDE to call the ARM command-line tool `fromelf` to display various 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 such as 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 disassemble library files selected in the project window. You can also disassemble output that has been processed by the `fromelf` utility. See *Disassembling fromelf output* on page 4-49 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 postlinker. See *Configuring target settings* on page 10-8 and *Configuring RealView fromelf* on page 10-75 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 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.

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. You can select both source and 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 format.

See the *RealView Compilation Tools Linker and Utilities Guide* for more information on using `fromelf`, including information on splitting `fromelf` output for multiple memory banks.

To configure `fromelf` to process output images you must:

1. Configure the Target settings panel to call `fromelf` as a postlinker. 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 10-75 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.

Disassembling fromelf output

You can configure the CodeWarrior IDE to call `fromelf` to:

- convert output to a different binary format
- disassemble the converted output.

To disassemble converted binary output, you must configure the CodeWarrior IDE to call `fromelf` twice:

1. Configure the Target settings 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 10-75 for detailed instructions.
3. Configure the **Debugger** or the **RealView Runner** panel to call `fromelf` as a third-party debugger to disassemble the converted output binary. See *Configuring the ARM Debuggers* on page 10-84 for detailed instructions.
4. Select **Run** or **Debug** from the **Project** menu, depending on whether you have configured the Runner or Debugger panel to call `fromelf`. See *Configuring the RealView Runner* on page 10-90 for more information on configuring a debugger to run your images. The CodeWarrior IDE compiles and links your code to produce an executable ELF output file, and then calls `fromelf` as a postlinker to convert the output to the binary format of your choice.

   When the compile and postlink operations are finished, the CodeWarrior IDE calls `fromelf` as a third party debugger with the command-line options you have specified in the configuration panel. Refer to the Utilities chapter of the *RealView Compilation Tools Linker and Utilities Guide* for detailed information on the command-line options to `fromelf`.

4.11.3 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 panel to call the ARM RealView Librarian as the linker. See *Configuring target settings* on page 10-8 for more information. 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 subproject, 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 Chapter 4 *Working with the ARM Project Stationery* for more information.

4.12 Running batch files with the batch runner

CodeWarrior for RVDS provides a batch runner utility that you can use to run a DOS batch file from your project. To use the batch runner you must:

- Configure the file mappings for your build target to recognize `.bat` files. By default, the ARM project stationery supplied with RVDS is configured to recognize batch files.
- Configure the batch runner as the postlinker in the Target Settings panel.

  Note

  You can configure only one postlinker. 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 make the build target. The batch runner is run only after a successful link operation.

This section includes:

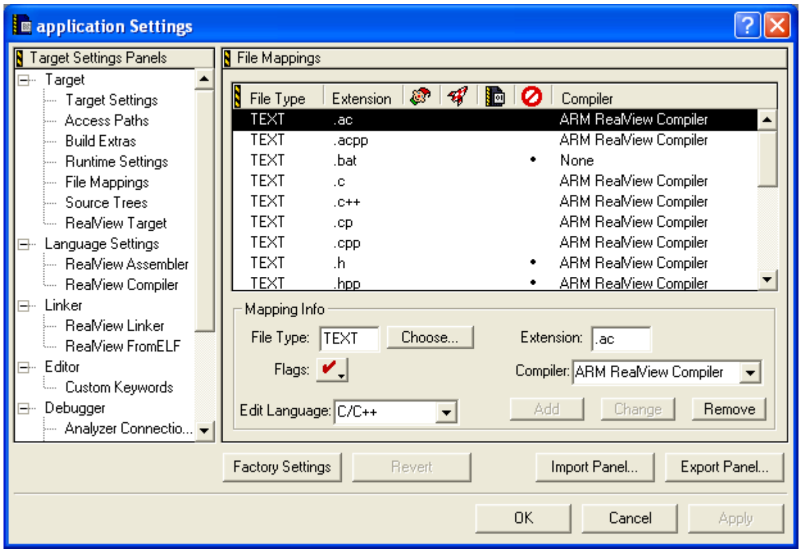
- *Configuring file mappings to recognize batch files*
- *Configuring the batch runner as the postlinker* on page 4-52
- *Adding batch files and making the build target* on page 4-53.

4.12.1 Configuring file mappings to recognize batch files

If you use the ARM project stationery supplied with RVDS to create a new project, then the `.bat` file mapping is already configured.

If you have a legacy project, then you must configure the file mappings for each build target to which you want to add batch files. Do this as follows:

1. Display the Target Settings panel for the build target you want to configure (see *Using the Target Settings window* on page 10-4).
2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel.
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 list and select **None**.
6. Click **Add** to add a new file mapping. See Figure 4-19 on page 4-52.



File Type	Extension		Compiler
TEXT	.ac		ARM RealView Compiler
TEXT	.acpp		ARM RealView Compiler
TEXT	.bat	•	None
TEXT	.c		ARM RealView Compiler
TEXT	.c++		ARM RealView Compiler
TEXT	.cp		ARM RealView Compiler
TEXT	.cpp		ARM RealView Compiler
TEXT	.h	•	ARM RealView Compiler
TEXT	.hpp	•	ARM RealView Compiler

File Type: TEXT
Extension: .ac
Flags: ✓
Compiler: ARM RealView Compiler
Edit Language: C/C++

Figure 4-19 Specifying batch file mappings

7. Click **Save** to save your changes.

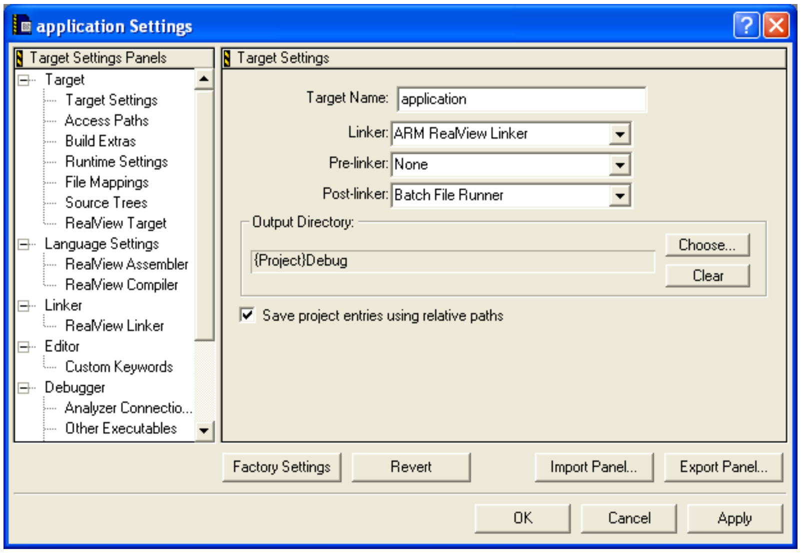
4.12.2 Configuring the batch runner as the postlinker

To run batch files from your project, you must configure the batch runner to be run as a postlinker in the Target Settings configuration panel.

To configure the batch runner:

1. Display the Target Settings panel for the build target you want to configure (see *Using the Target Settings window* on page 10-4).
2. Click **Target Settings** in the Target Settings Panels list to display the configuration panel (Figure 4-20 on page 4-53).
3. Click the **Post-linker** drop-down list and select **Batch File Runner**. See Figure 4-20 on page 4-53.

   See *Using the Target Settings window* on page 10-4 for more information on the other options on this panel.


application Settings

Target Settings Panels
- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - Runtime Settings
  - File Mappings
  - Source Trees
  - RealView Target
- Language Settings
  - RealView Assembler
  - RealView Compiler
- Linker
  - RealView Linker
- Editor
  - Custom Keywords
- Debugger
  - Analyzer Connectio...
  - Other Executables

Target Settings

Target Name: application

Linker: ARM RealView Linker

Pre-linker: None

Post-linker: Batch File Runner

Output Directory: {Project}Debug

☑ Save project entries using relative paths

Figure 4-20 Target Settings panel

4. Click **Save** to save 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 you have configured to accept files with a `.bat` extension. You can add any number of batch files to the build target, however 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 you have not configured to accept `.bat` file extensions, the CodeWarrior for RVDS displays a warning message and adds the batch files only to the properly configured targets.

————————

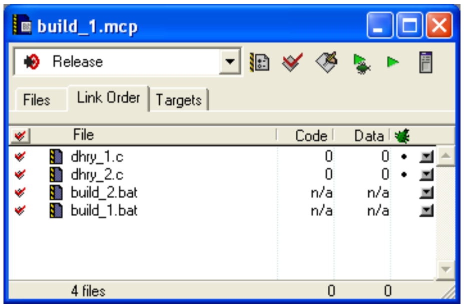
To add the batch files and make your build target:

1. Either:
   - select **Add Files…** from the **Project** menu
   - drag and drop the batch files onto the project window.

   See the example in *Adding source files to the build targets* on page 4-13 for more information.

2. Touch your project files, if required, to ensure that the project is rebuilt. 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. Click the **Link Order** tab to specify which batch file is to be run, if you have added more than one batch file to the build target. Drag the batch file you want to execute so that it is displayed before any other batch file (Figure 4-21).



File	Code	Data
dhry_1.c	0	0
dhry_2.c	0	0
build_2.bat	n/a	n/a
build_1.bat	n/a	n/a
4 files	0	0

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, `build_2.bat` in Figure 4-21 is executed first.

Chapter 5
Working with the ARM Debuggers

This chapter describes how to use the CodeWarrior IDE and the ARM debuggers to run and debug your code. It contains the following sections:

5.1 About working with the ARM debuggers

You can call any of the ARM debuggers from the CodeWarrior IDE to either debug, or run images output from a make operation. This section gives an overview of how the ARM debuggers integrate 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 ARM debuggers work with the CodeWarrior IDE

The CodeWarrior IDE for RVDS enables you to call an ARM Debugger with a number of optional arguments, depending on the debugger you are using. When you select **Debug** or **Run** from the **Project** menu, the CodeWarrior IDE starts your selected debugger and instructs it to load the image file output from the current build target. 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.

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.

Selecting the ARM debugger and ARM runner

You can select any of the ARM debuggers to either run or debug your executable images. You do not have to use the same debugger for running and debugging. The following debuggers are available:

- RealView Debugger
- *ARM eXtended Debugger* (AXD)
- *ARM symbolic debugger* (`armsd`).

  Note

  AXD and `armsd` are supported for ARM7 and ARM9 targets only.

In addition, you can select a third-party debugger in place of the ARM debuggers. See *Configuring the debugger* on page 10-83 for detailed information on selecting a debugger and a runner.

See the *RealView Debugger Essentials Guide* for detailed information on using the ARM debuggers.

Selecting debug options

There are a number of options to the RealView compiler that affect the quality of the debug view available to the debuggers. 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. See *Using the Debug and Release build targets* for more information. See *Configuring the RealView compiler* on page 10-47 for more information on setting build options yourself.

Using the Debug and Release build targets

The default project stationery provided with CodeWarrior IDE for RVDS defines two build targets for each project type:

Debug This build target is configured to generate the most complete debug information possible for each source file in the build target.

Release This build target is configured to generate the highest level of code optimization at the expense of debugging information.

See Chapter 4 *Working with the ARM Project Stationery* for more information on the default build targets. See also *Working with multiple build targets and subprojects* on page 3-44 for more information on using multiple build targets in your projects.

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. See *Generating debug information for individual source files* for more information.
- Use the Compiler and 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. See *Generating debug information for all source files in a build target* on page 5-6 for more information.

  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.

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 the ARM tools are not configured to generate debug information for the entire build target. See *Generating debug information for all source files in a build target* on page 5-6 for more information. 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 3-39 for more information.

To generate debugging information for a source code file:

1. Select the build target for which you want to generate debug information. See *Setting the current build target* on page 3-46 for more information.

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.

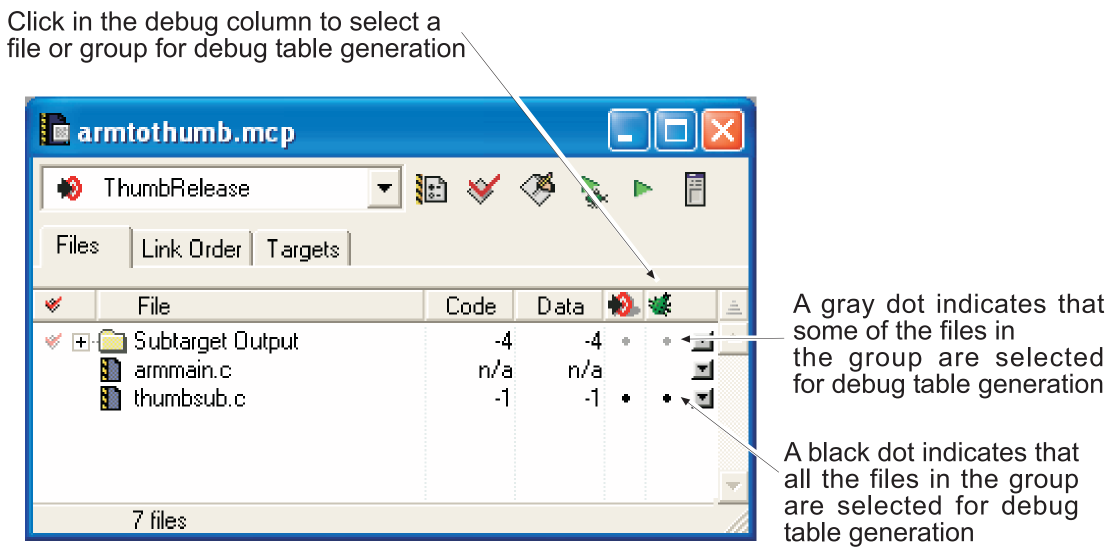
     Note

     To generate debug information for source files in a subtarget or subproject you must open the build target or subproject.

   For selected files, the debug column displays a Debug Info marker (Figure 5-1 on page 5-6) and marks the source files for recompilation.

   For selected groups, the Debug column displays one of three markers:

   - a black marker indicates that all source files in the group generate debugging information
   - a gray marker indicates that only some of the source files in the group generate debugging information
   - no marker indicates that no debugging information is generated for source files in the group.


Click in the debug column to select a file or group for debug table generation
armtothumb.mcp
ThumbRelease
Files
Link Order
Targets
File
Code
Data
Subtarget Output
armmain.c
thumbsub.c
7 files
A gray dot indicates that some of the files in the group are selected for debug table generation
A black dot indicates that all the files in the group are selected for debug table generation

File	Code	Data		
Subtarget Output	-4	-4	•	•
armmain.c	n/a	n/a		
thumbsub.c	-1	-1	•	•

Figure 5-1 Debug Info markers

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, provided the tool configuration panels are not configured to generate debug information.

5.2.2 Generating debug information for all source files in a build target

When you click on 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.

See *Generating debug information* on page 5-4 for more information.

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 for the Realview assembler.

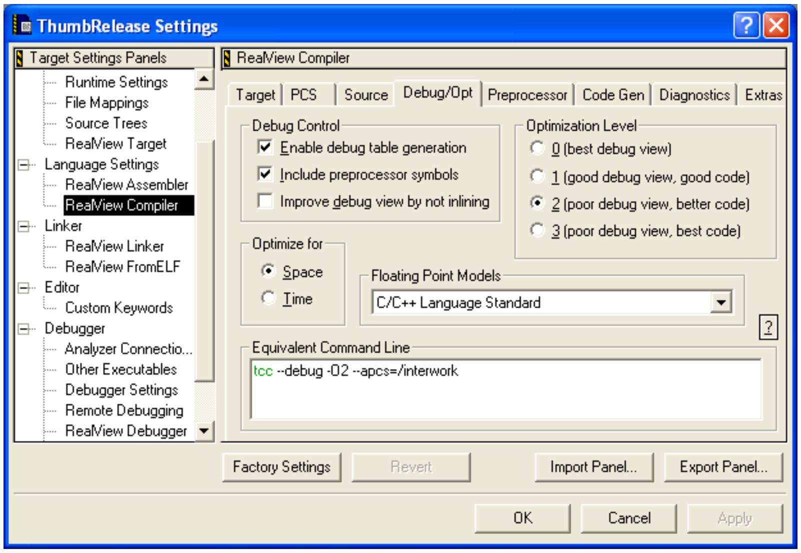
Note

The default ARM project stationery is configured to generate debug information for all source files in the Debug build target. See *Using the Debug and Release build targets* on page 5-3 for more information.

Example: Using the compiler configuration dialogs

The following example shows how to enable debug setting for the RealView compiler. The steps 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 4-39 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 the RealView Compiler entry in the Target Settings Panels list, and click the **Debug/Opt** tab to display the configuration panel (Figure 5-2).


ThumbRelease Settings

Target Settings Panels: Runtime Settings, File Mappings, Source Trees, RealView Target, Language Settings, RealView Assembler, RealView Compiler, Linker, RealView Linker, RealView FromELF, Editor, Custom Keywords, Debugger, Analyzer Connectio..., Other Executables, Debugger Settings, Remote Debugging, RealView Debugger

RealView Compiler

Target | PCS | Source | Debug/Opt | Preprocessor | Code Gen | Diagnostics | Extras

Debug Control
- ✓ Enable debug table generation
- ✓ Include preprocessor symbols
- ☐ Improve debug view by not inlining

Optimization Level
- 0 (best debug view)
- 1 (good debug view, good code)
- [✓ 2 (poor debug view, better code)]
- 3 (poor debug view, best code)

Optimize for
- [✓ Space]
- Time

Floating Point Models: C/C++ Language Standard

Equivalent Command Line

```
tcc --debug -O2 --apcs=/interwork
```

Figure 5-2 RealView compiler Debug/Opt panel

4. Select **Enable debug table generation** to instruct the compiler to generate DWARF2 debug tables. See *Configuring debug and optimization* on page 10-55 for more information on the other options available on this panel. See *Configuring miscellaneous assembler options* on page 10-41 for detailed information on selecting Assembler debug options.

5. Click **Save** to save your changes. 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.

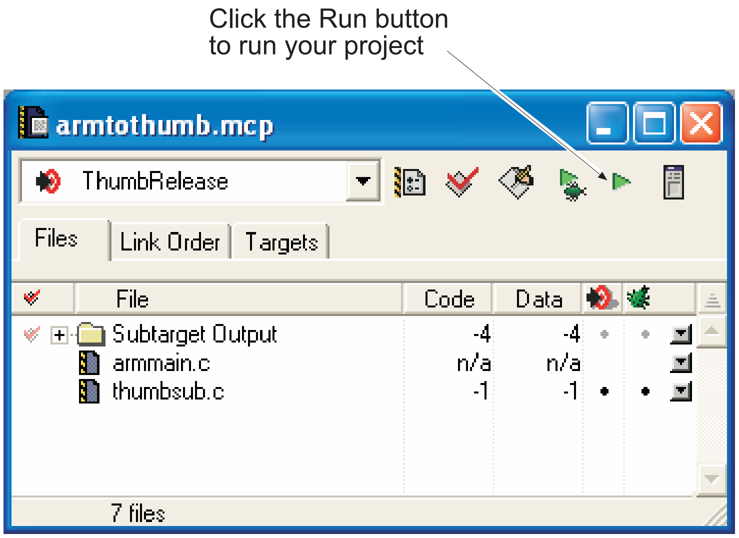
5.3 Running and debugging your code

This section describes how to run executable images from within the CodeWarrior IDE, and how to call one of the ARM debuggers to run or debug your code.

5.3.1 Running a project

To call an ARM 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, or click the **Run** button (Figure 5-3).



File	Code	Data
Subtarget Output	-4	-4
armmain.c	n/a	n/a
thumbsub.c	-1	-1

Figure 5-3 The Run button

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 target configuration panel (see *Configuring the RealView Runner* on page 10-90).

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 an ARM 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 set the correct debug options for your build target. See:
   - *Configuring the ARM Debuggers* on page 10-84 for information on how to select an ARM debugger.
   - *Configuring assembler and compiler language settings* on page 10-35 for information on how to enable debug table generation for the 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.

   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 *Configuring the ARM Debuggers* on page 10-84).

See the *RealView Debugger Essentials Guide* for detailed information on using the ARM debuggers.

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:

- the Errors & Warnings message window
- the Notes message window.

These are described in:

- *Overview of the message window*
- *Using the message window* on page 5-14.

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:

Error & Warnings message window

This message window displays error and warning messages from the compiler, assembler, linker, and post-linker.

Notes message window

This message window displays all other types of message, including:

- The results of a batch Find operation.
- 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 preferences panel. See *Configuring build settings* on page 9-6 for more information.

  Error and warning messages are not displayed in the Notes message window.

The message window contains interface elements that enable you 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.

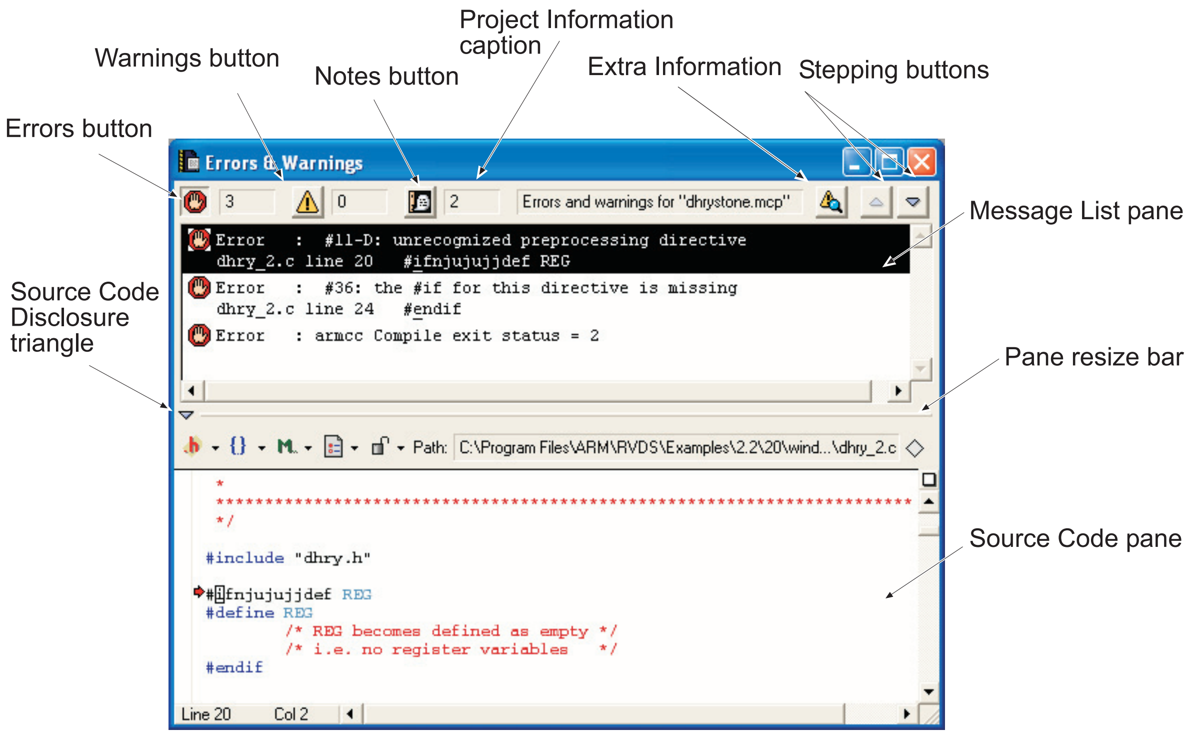
Some user interface items in the message window are not described here. See *Overview of the editor window* on page 6-3 for more information on:

- the **Markers** drop-down menu
- the **Document Settings** drop-down menu
- the **Version Control** drop-down menu
- the **Line Number** button
- the File Path caption.

Figure 5-4 on page 5-13 shows an example of the Errors & Warnings message window. The major interface components of the window are described below.

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.


Project Information caption
Warnings button
Notes button
Extra Information
Stepping buttons
Errors button
Message List pane
Source Code Disclosure triangle
Pane resize bar
Source Code pane
Errors & Warnings
3
0
2
Errors and warnings for "dhrystone.mcp"
Path: C:\Program Files\ARM\RVDS\Examples\2.2\20\wind...\dhry_2.c
Line 20
Col 2

```
Error   :  #11-D: unrecognized preprocessing directive
dhry_2.c line 20   #ifnjujujjdef REG
Error   :  #36: the #if for this directive is missing
dhry_2.c line 24   #endif
Error   : armcc Compile exit status = 2
```

```
 *
 ***********************************************************************
 */

#include "dhry.h"

#ifnjujujjdef REG
#define REG
        /* REG becomes defined as empty */
        /* i.e. no register variables   */
#endif
```

Figure 5-4 The Errors & Warnings message window

The major interface components of the 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 5-15 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 5-15 for more information.

Warnings button

The Warnings button toggles the view of warning messages on and off. See *Viewing error and warning messages* on page 5-15 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 5-16 for more information.

Message List pane

The Message List pane displays your messages. See *Viewing error and warning messages* on page 5-15 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 5-15 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.

5.4.2 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* on page 5-15
- *Filtering error and warning messages* on page 5-15
- *Stepping through messages* on page 5-16

- *Correcting compilation errors and warnings* on page 5-16
- *Correcting link errors* on page 5-17
- *Printing the message window* on page 5-18
- *Saving the message window* on page 5-18.

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

You can choose whether the Errors & Warnings message window displays either error messages, or warning messages, or both by using the **Errors** button and the **Warnings** button at the top of the message window.

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.
- To view a combination of message types, click all the required buttons and ensure that all other buttons are deselected.

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. See the following sections for information on how to correct errors and warnings.

Correcting compilation errors and warnings

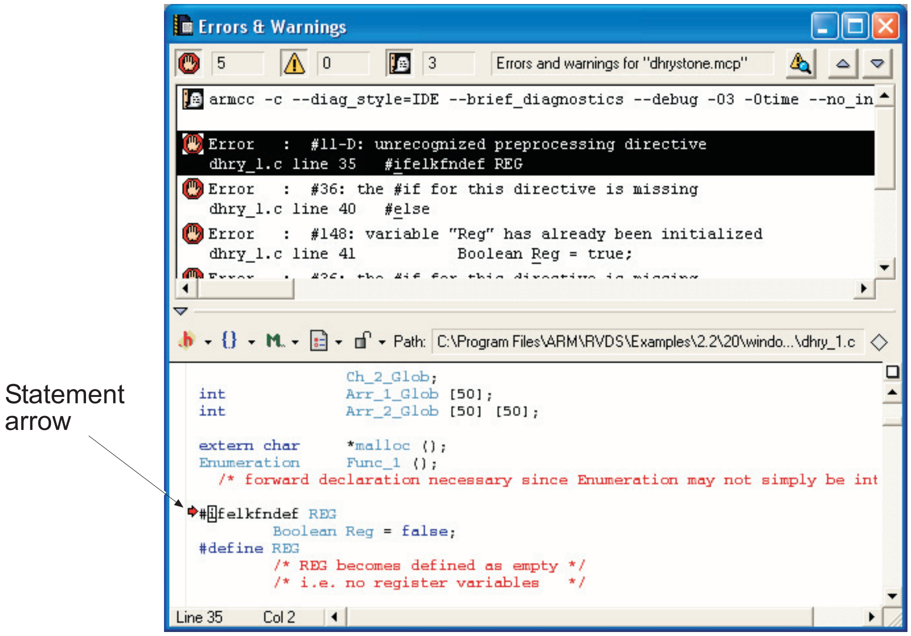
To correct a compilation error or warning from the Source Code pane of the message window:

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 5-4 on page 5-13).

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. (Figure 5-5 on page 5-17).

   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.


Statement arrow
Errors & Warnings
5
0
3
Errors and warnings for "dhrystone.mcp"
Path: C:\Program Files\ARM\RVDS\Examples\2.2\20\windo...\dhry_1.c
Line 35
Col 2

```
armcc -c --diag_style=IDE --brief_diagnostics --debug -O3 -Otime --no_in

Error   :  #11-D: unrecognized preprocessing directive
dhry_1.c line 35   #ifelkfndef REG
Error   :  #36: the #if for this directive is missing
dhry_1.c line 40   #else
Error   :  #148: variable "Reg" has already been initialized
dhry_1.c line 41           Boolean Reg = true;
```

```
                        Ch_2_Glob;
int                     Arr_1_Glob [50];
int                     Arr_2_Glob [50] [50];

extern char             *malloc ();
Enumeration             Func_1 ();
  /* forward declaration necessary since Enumeration may not simply be int

#ifelkfndef REG
            Boolean Reg = false;
#define REG
            /* REG becomes defined as empty */
            /* i.e. no register variables   */
```

Figure 5-5 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 assembler PCS options* on page 10-39 and *Configuring compiler PCS options* on page 10-51 for more information.
- 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 use the RVCT22LIB environment variable to search for libraries. See *Configuring linker options* on page 10-68 for more information.
- You have not correctly linked the output from a dependent build target or subproject. You must explicitly specify that output from a subproject or subtarget is linked into your final output image. See *Working with multiple build targets and subprojects* on page 3-44 for more information.
- You have not correctly set up your linker output options. See *Configuring the RealView linker* on page 10-65 for more information.

Link errors are displayed in the message window in the same way as compilation errors. See *Viewing error and warning messages* on page 5-15 for information on how to move through the message window.

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 message window is printed.

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 contents of the active message window to a text file.

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 entire contents of the message window is copied to the clipboard. You can paste the clipboard contents into a text editor or other application.

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:

6.1 About editing source code

The CodeWarrior editor is a full-featured text editor designed for programmers. Its features include:

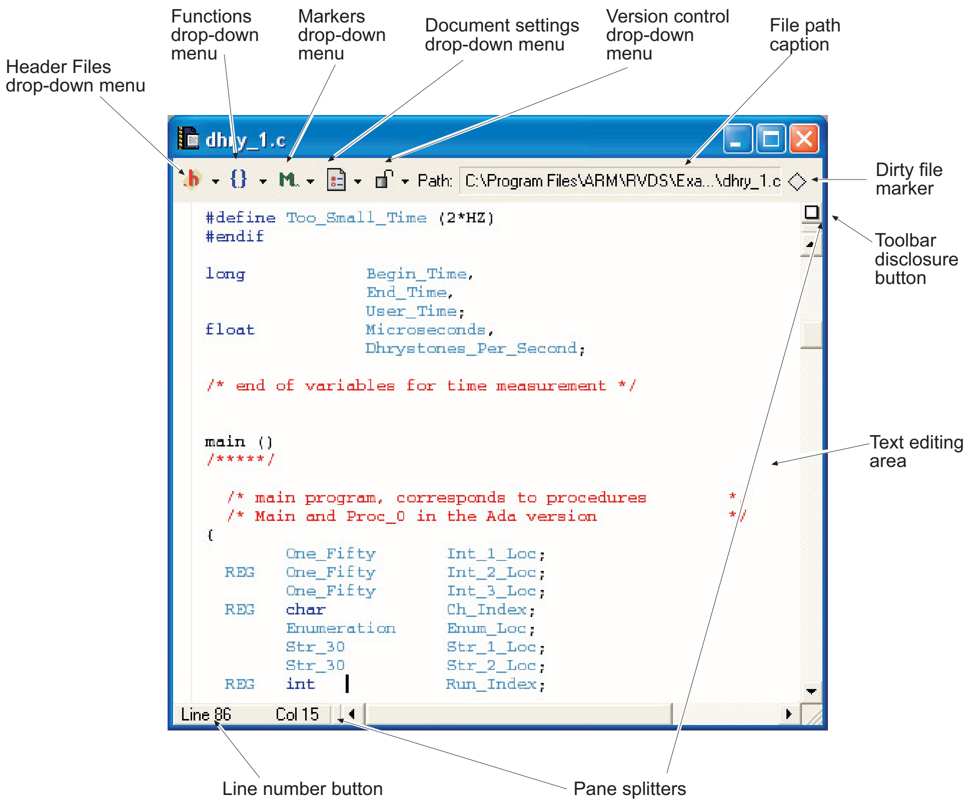
- drop-down menus on every editor window for opening header files and quickly navigating among functions.
- integrated VCS (Version Control System) menus that enable you to work with your VCS from within the CodeWarrior IDE. (This menu is only available if a version control system plug-in module has 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 editor. You can customize the way the CodeWarrior editor works. See *Editor settings* on page 9-23 for more information. 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 source code browser.

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 editor window.


Header Files drop-down menu
Functions drop-down menu
Markers drop-down menu
Document settings drop-down menu
Version control drop-down menu
File path caption
Dirty file marker
Toolbar disclosure button
Text editing area
dhry_1.c
Path: C:\Program Files\ARM\RVDS\Exa...\dhry_1.c
Line 86
Col 15
Line number button
Pane splitters

```
#define Too_Small_Time (2*HZ)
#endif

long            Begin_Time,
                End_Time,
                User_Time;
float           Microseconds,
                Dhrystones_Per_Second;

/* end of variables for time measurement */


main ()
/*****/

  /* main program, corresponds to procedures        */
  /* Main and Proc_0 in the Ada version             */
{
        One_Fifty       Int_1_Loc;
  REG   One_Fifty       Int_2_Loc;
        One_Fifty       Int_3_Loc;
  REG   char            Ch_Index;
        Enumeration     Enum_Loc;
        Str_30          Str_1_Loc;
        Str_30          Str_2_Loc;
  REG   int             Run_Index;
```

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 (Figure 6-2 on page 6-4) to either:

- open header files referenced by the current file
- use the **Touch** and **Untouch** commands from this drop-down menu.

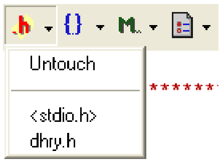


Figure 6-2 Header files drop-down menu

See *Touching and untouching files* on page 3-38 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. See *Using the Functions drop-down menu* on page 6-19 for more information.

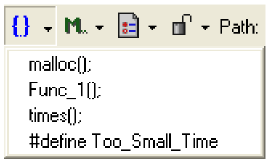


Figure 6-3 Functions drop-down menu

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.

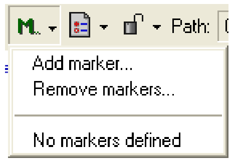


Figure 6-4 The Marker drop-down menu

See *Using markers* on page 6-21 for more information.

Document Settings drop-down menu

You can use the **Document Settings** drop-down menu (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.

See *Text Colors* on page 9-31 for details of how to modify syntax coloring.

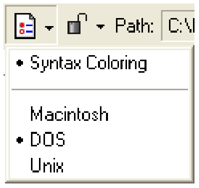


Figure 6-5 The Document Settings drop-down menu

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, at the top right of the window shown in Figure 6-1 on page 6-3.

Dirty File marker

The Dirty File marker indicates if the file displayed in a window has been modified after it was last saved or opened. 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. Figure 6-7 on page 6-8 shows an editor window with multiple panes.

See *Splitting the window into panes* on page 6-8 for more information on pane splitter controls.

Line Number button

The Line Number button shown in Figure 6-1 on page 6-3 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.

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 the following 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.

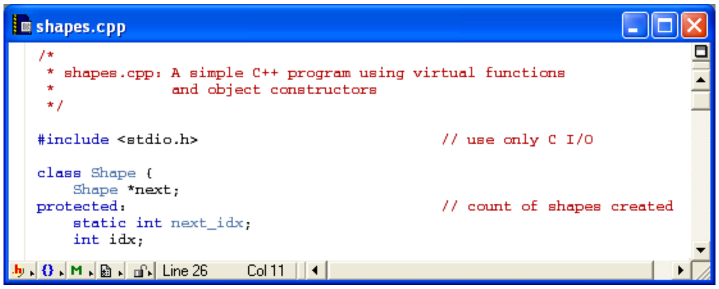
6.3.1 Setting text size and font

Use the Font & Tabs preference panel to set the size or font used to display text in an editor window. See *Font & Tabs* on page 9-28 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. Use the **Toolbar Disclosure** button, shown in Figure 6-1 on page 6-3, to show or hide the toolbar.

To hide the toolbar, click the **Toolbar Disclosure** button. The CodeWarrior IDE hides the toolbar, and displays the default toolbar drop-down menu controls along the bottom of the editor window.



```
/*
 * shapes.cpp: A simple C++ program using virtual functions
 *             and object constructors
 */

#include <stdio.h>                              // use only C I/O

class Shape {
    Shape *next;
protected:                                      // count of shapes created
    static int next_idx;
    int idx;
```

Figure 6-6 Toolbar at bottom of Editor window

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.

- The File Path caption is no longer visible when the toolbar is hidden.

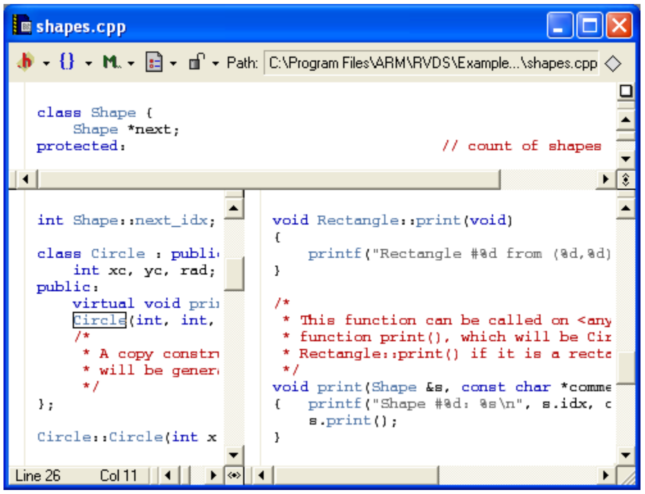
To re-show the toolbar if it is hidden, click the **Toolbar Disclosure** button. The toolbar is displayed along the top of the editor window.

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. The following sections describe how to create, resize, and remove multiple panes.



```
class Shape {
    Shape *next;
protected:                              // count of shapes
```

```
int Shape::next_idx;

class Circle : publi
    int xc, yc, rad;
public:
    virtual void pri
    Circle(int, int,
    /*
     * A copy constr
     * will be gener
     */
};

Circle::Circle(int x
```

```
void Rectangle::print(void)
{
    printf("Rectangle #%d from (%d,%d)
}

/*
 * This function can be called on <any
 * function print(), which will be Cir
 * Rectangle::print() if it is a recta
 */
void print(Shape &s, const char *comme
{   printf("Shape #%d: %s\n", s.idx, c
    s.print();
}
```

Figure 6-7 Multiple panes in a window

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 (the top and left side) of a pane in the editor window. 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 line 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 line 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
- when 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.

6.4 Editing text

The CodeWarrior IDE provides many methods for editing source files. These methods 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 6-15
- *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

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

To add text to an open file:

1. Click once in the text editing area of the window to set the new location of the text insertion point.
2. Begin typing on the keyboard to enter text.

See *Basic editor window navigation* on page 6-10 for ways to move the insertion point in an editor window.

Also, see for information on helping you to ensure consistency in the text that you enter.

Deleting text

You can delete text in any of the following ways:

- press the Backspace key to delete text that is behind the text insertion point
- press the Delete key to delete text that is in front of the text insertion point
- select the text you want to delete and press the Backspace or Delete key to delete the selection. See *Selecting text* on page 6-12, below, for details on how to select text.

Using cut, copy, paste, and clear

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 more information on these commands.

6.4.3 Selecting text

There are several ways to select text in the editor window. This section describes how to select text:

- using keystroke shortcuts
- using the mouse.

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

Select text to...	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.

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 preference 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 preference 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 IDE configuration panels. See *Editor settings* on page 9-23 for information on how to turn this feature on or off.

The CodeWarrior editor accepts drag-and-drop text items from other applications that support drag-and-drop editing.

Moving text in the text editing area

To drag and drop text between text areas in the CodeWarrior IDE:

1. Either:
   - create a new text file
   - open an existing text file.

   See *Creating and opening files* on page 2-3 for more information.

2. 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:

1. Place the insertion point in the text you want to check.

2. Select **Balance** from the **Edit** menu. Alternatively, double-click on a parenthesis, bracket, or brace character that you want to check for a matching character.

   The CodeWarrior 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.

See also *Formatting code* on page 6-15 and *Completing code* on page 6-15 for more information on how to configure the CodeWarrior IDE to format and complete code automatically as you type.

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 IDE configuration panels. See *Code formatting* on page 9-21 for information on how to enable this feature.

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.

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. For a complete list of the code formatting options available and their effect, see *Code formatting* on page 9-21.

6.4.7 Completing code

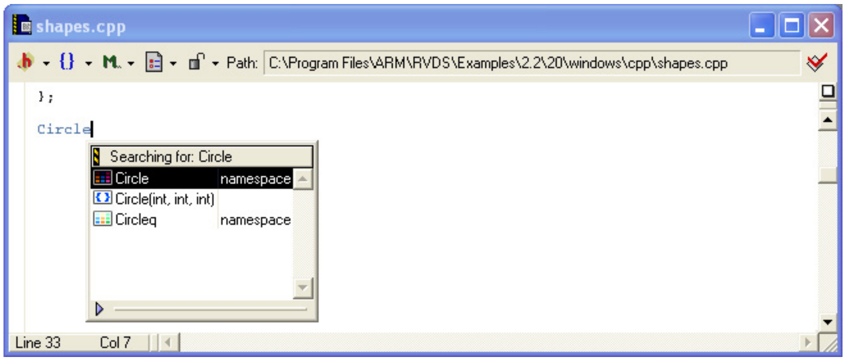
The Code Warrior IDE provides code completion options for automatically suggesting ways to complete source code as it is entered in the Editor window. By using code completion you avoid referring to other files to remember available symbols, thereby reducing the potential for introducing errors in your code. To use code completion, you must enable this feature in the IDE configuration panels. See *Code completion* on page 9-20 for information on how to enable this feature.

C/C++ Code Completion will function more effectively when **Language Parser** is selected for the **Generate Browser Data From** option in the **Build Extras** setting. CodeWarrior is then able to use the database of symbols defined in the browser to provide code completion suggestions. See *Configuring build extras* on page 10-20.

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. To complete the source code:
   - Choose **Complete Code** from the **Edit** menu.
   - Type Alt- . (Alt-period).

The Code Completion window is displayed beneath the source code containing suggestions for completing the code. See Figure 6-8 on page 6-16.



```
};

Circle
```

Figure 6-8 Code completion window

3. Use the information displayed in the Code Completion window to help you complete the code at the insertion point. 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.

   To select the next code completion item, type Alt-/. To select the previous code completion item, type Alt-Shift-/.

You can also use the following commands to cycle-through code-completion suggestions in the Editor window:

- To display the next code completion item choose **Get Next Completion** from the **Edit** menu (or type Alt-/).
- To display the previous code completion item choose **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 9-20 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 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 9-28 for information on configuring the number of spaces defined for a tab character.

6.4.9 Undoing changes

The CodeWarrior 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 the file.

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 just 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 Configuration panel. See *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 ‘uncuts’ 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 will 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.

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 9-31 for information on configuring color syntax options.

6.5 Navigating text

The CodeWarrior 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
- *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 6-24.

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 editor. See *Customizing keybindings* on page 9-43 for more information on how to the 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 (Figure 6-9 on page 6-20). The drop-down menu lists the functions in your source file.

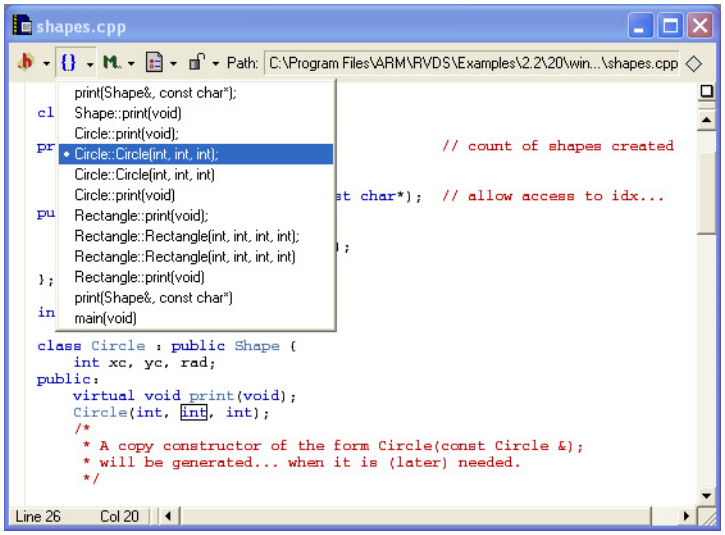
By default, the menu lists the functions in the order in which they appear in the file. You can list functions in alphabetical order by pressing the control key before you click on the **Functions** drop-down menu.

——— **Note** ———

You can change the default order of the functions in the **Functions** drop-down menu with the *Sort Function Popup* configuration option. See *Editor settings* on page 9-23 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 for that function. (See Figure 6-9.)


shapes.cpp
Path: C:\Program Files\ARM\RVDS\Examples\2.2\20\win...\shapes.cpp
print(Shape&, const char*);
Shape::print(void)
Circle::print(void);
• Circle::Circle(int, int, int);
Circle::Circle(int, int, int)
Circle::print(void)
Rectangle::print(void);
Rectangle::Rectangle(int, int, int, int);
Rectangle::Rectangle(int, int, int, int)
Rectangle::print(void)
print(Shape&, const char*)
main(void)
Line 26
Col 20

```
cl
pr                                                   // count of shapes created

                                         st char*);  // allow access to idx...
pu
                                         );

};
in
class Circle : public Shape {
    int xc, yc, rad;
public:
    virtual void print(void);
    Circle(int, int, int);
    /*
     * A copy constructor of the form Circle(const Circle &);
     * will be generated... when it is (later) needed.
     */
```

Figure 6-9 The Functions drop-down menu with selected function

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.

If CodeWarrior 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 8-25). Otherwise, the CodeWarrior IDE displays a message window (see *Using the message window* on page 5-14).

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 editor. Markers set places in your file that you can name and jump to quickly, or for leaving notes to yourself about work in progress on your code. You can also use bookmarks to set places in your file. See

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).

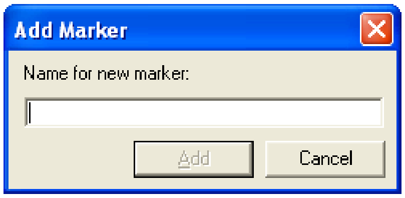


Figure 6-10 Add Marker dialog box

3. Enter text in the dialog box to mark your insertion point location in the file with a note, comment, function name, or other text that would be helpful to you.
4. Click **Add**. Your marker will be visible in the **Markers** drop-down menu (Figure 6-11 on page 6-22).

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. This is useful for quickly adding specific functions or lines as markers.

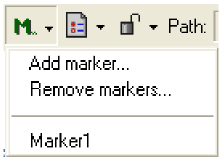


Figure 6-11 Example text file with a marker added

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).

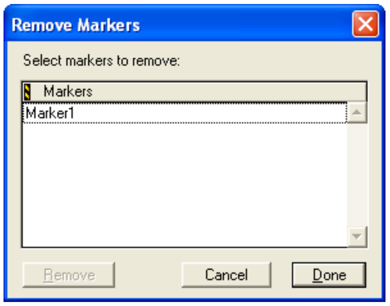


Figure 6-12 Remove Markers dialog box

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.

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

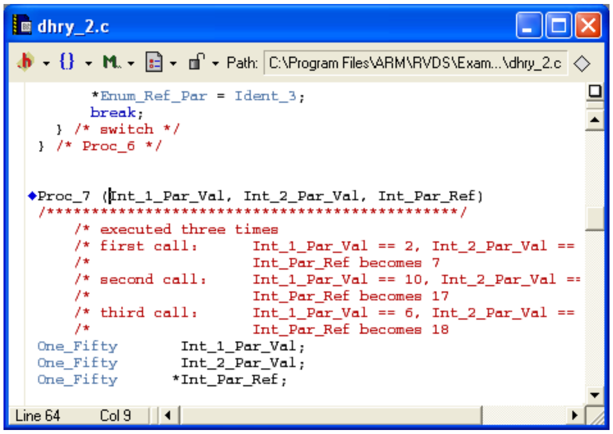
You can add or remove bookmarks in any of your text files using the facilities built-in to the CodeWarrior editor. Bookmarks are similar to markers as they set places in the code you can jump to but they differ from markers as they cannot be given a name. They provide a quick way to set a place in the file.

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).



```
        *Enum_Ref_Par = Ident_3;
        break;
    } /* switch */
  } /* Proc_6 */


◆Proc_7 (Int_1_Par_Val, Int_2_Par_Val, Int_Par_Ref)
 /**************************************************/
     /* executed three times
     /* first call:       Int_1_Par_Val == 2, Int_2_Par_Val ==
     /*                   Int_Par_Ref becomes 7
     /* second call:      Int_1_Par_Val == 10, Int_2_Par_Val ==
     /*                   Int_Par_Ref becomes 17
     /* third call:       Int_1_Par_Val == 6, Int_2_Par_Val ==
     /*                   Int_Par_Ref becomes 18
 One_Fifty        Int_1_Par_Val;
 One_Fifty        Int_2_Par_Val;
 One_Fifty       *Int_Par_Ref;
```

Figure 6-13 Editor window containing a bookmark

To remove a bookmark

To remove a bookmark, move the text insertion point to line containing the bookmark and type Ctrl-F2. The bookmark is removed.

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

Line: 60

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. Click **Line Number** on the editor window to open the Line Number dialog box (Figure 6-14).

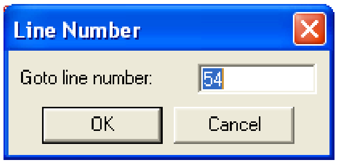


Figure 6-14 Line Number dialog box

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 when you use 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. See Chapter 8 *Working with the Browser* for more information on using the browser.

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.

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 use the **Touch** and **Untouch** commands from this menu. See *Touching and untouching files* on page 3-38 for more details.

———————————

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 list of files in the **Header Files** menu is not displayed.
2. In the editor window for the your source file, click the **Header Files** drop-down menu icon to display the menu (Figure 6-15).

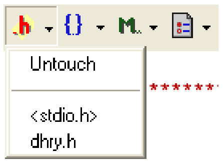


Figure 6-15 The Header Files drop-down menu

Note

- You can also open the Header Files drop-down menu from the project window.
- Some files cannot be opened in the editor window, such as libraries.

3. Select the header file you want to open from the menu. The header file is opened in a new editor window.

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.

Chapter 7
Searching and Replacing Text

This chapter describes how to use the CodeWarrior IDE search and replace functions. It contains the following sections:

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 search for and replace text in a single file, in every file in a project, or in any combination of files. You can also search using regular expressions, similar to the UNIX `grep` command.

The Find dialog, shown in Figure 7-1 on page 7-4, provides comprehensive search and replace facilities. You can use the Find dialog 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.

7.2 Finding and replacing text in a single file

The Find dialog box enables you 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.

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* 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.

When you search for selected text, the CodeWarrior IDE uses the option settings that you last chose in the Find dialog box. To change the option settings, 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 backwards 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.

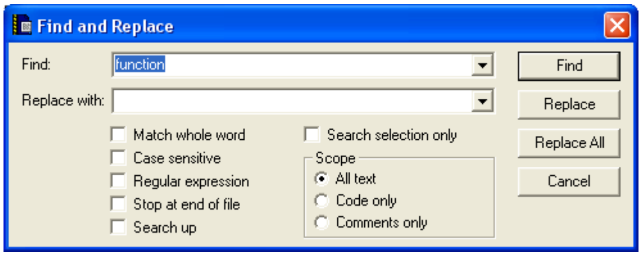
4. Either:

   - 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

To find and replace text with the **Find and Replace** dialog box:

1. Select **Replace…** or **Find…** from the **Search** menu. The Find (and Replace) dialog box is displayed (Figure 7-1).



Find: function
Replace with:
☐ Match whole word
☐ Case sensitive
☐ Regular expression
☐ Stop at end of file
☐ Search up
☐ Search selection only
Scope: [✓ All text] Code only Comments only

Figure 7-1 The 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 list (in the **Find** text field).

   The **Recent Strings** drop-down list contains strings that have previously been used for searches. Select an item in one of these drop-down lists 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 7-7 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 list (in the **Replace** text field), if you want to replace the found string.

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.

   Search selection only

   Select this option to instruct the CodeWarrior IDE to search only the currently selected text.

   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.

   Scope

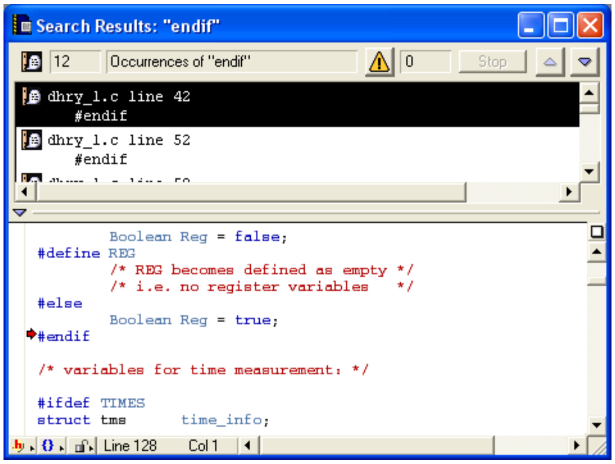
   Select an option from this group box to define the scope of the search. This can be one of:

   - 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.

- 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 (Figure 7-2).



```
dhry_1.c line 42
    #endif
dhry_1.c line 52
    #endif
```

```
        Boolean Reg = false;
#define REG
        /* REG becomes defined as empty */
        /* i.e. no register variables   */
#else
        Boolean Reg = true;
#endif

/* variables for time measurement: */

#ifdef TIMES
struct tms      time_info;
```

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 5-11 for more information on the features of the message window.

- 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 total 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 Find and Replace 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 or Replace text field
- enable the Regular expression option, and enter `\t` for Tab or `\r` for Return into the field.

Using regular expressions alters the way in which the CodeWarrior IDE locates a string match. See *Using grep-style regular expressions* on page 7-17 for more details.

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:

- folders
- 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 multi-file search*
- *Using file sets* on page 7-14.

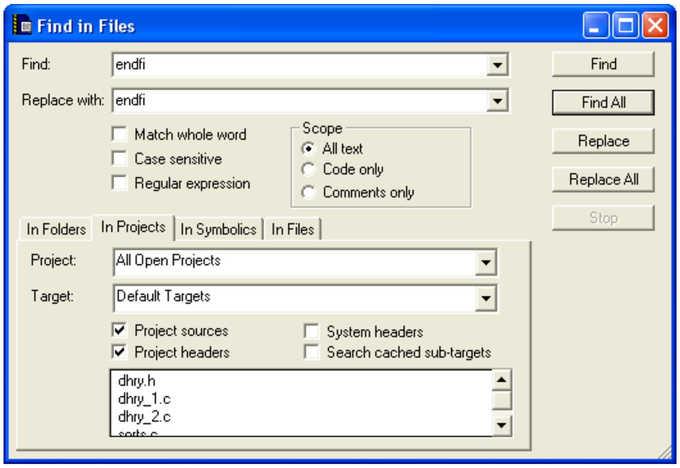
Note

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 multi-file search

This section describes how to perform multi-file searches. See *Using file sets* on page 7-14 for more information on configuring multi-file 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 *The Find in Files dialog* on page 7-9).



Find: endfi
Replace with: endfi
☐ Match whole word
☐ Case sensitive
☐ Regular expression
Scope: (● All text) ○ Code only ○ Comments only
Project: All Open Projects
Target: Default Targets
☑ Project sources
☑ Project headers
☐ System headers
☐ Search cached sub-targets
dhry.h
dhry_1.c
dhry_2.c
sorts.c

Figure 7-3 The Find in Files dialog

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 folders* 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* on page 7-13).
4. Type the search text into the Find text box.
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 Find All section in *Finding and replacing text with the Find and Replace dialog* on page 7-4 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.

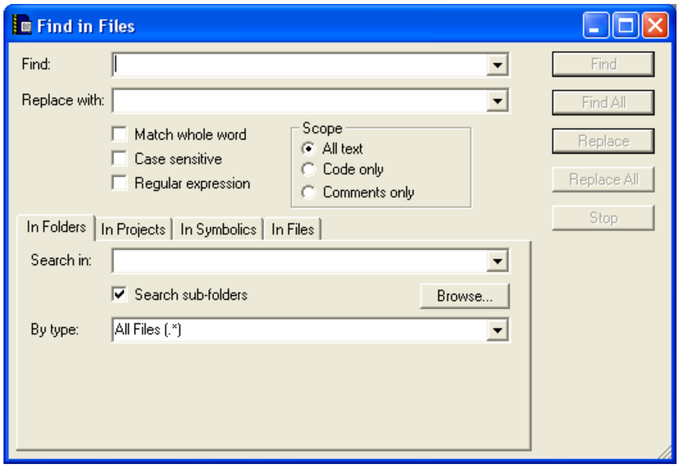
   Caution

   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 folders

To search multiple files by folders:

1. Select the **In Folders** tab in the Find in Files window.



Match whole word [ ], Case sensitive [ ], Regular expression [ ]; Scope: [✓ All text], Code only, Comments only; Search sub-folders [✓]; By type: All Files (.*)

Figure 7-4 The In Folders tab in the Find in Files Window

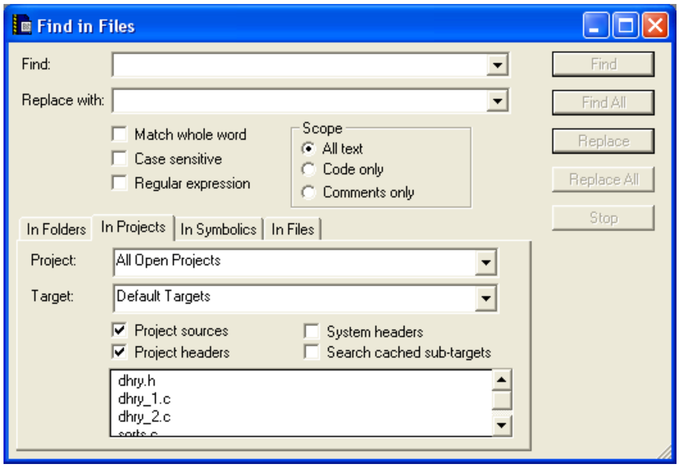
2. Use the **Search in** drop-down list to specify the location you want to search. Type in a location or click **Browse...** to find the required location.
3. Check **Search sub-folders** if you want to search in folders within the specified folder. If you leave this option unchecked, CodeWarrior IDE searches only files at this level.

4. Use the **By type** drop-down list 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. Select the **In Projects** tab in the **Find in Files** window.



Find:
Replace with:
☐ Match whole word
☐ Case sensitive
☐ Regular expression
Scope: [✓ All text] Code only Comments only
Project: All Open Projects
Target: Default Targets
☑ Project sources
☑ Project headers
☐ System headers
☐ Search cached sub-targets
dhry.h
dhry_1.c
dhry_2.c
sorts.c

Figure 7-5 The In Projects tab in the Find in Files Window

2. Use the **Project** drop-down list to specify the project you want to search.
3. Use the **Target** drop-down list to specify the target you want to search. Options are:
   - All Targets
   - Release
   - Debug.
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 4-44 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 4-44 for more information.

Search cached sub-targets

Select this option to search cached subtargets of the projects shown in the **Project** drop-down list. The search criteria that you define for the projects also applies to the cached subtargets. If this option is disabled, cached subtargets are disregarded during the search.

A cached subtarget has the following characteristics:

- the current target builds against it
- it resides in the current project
- it resides in a subproject if you enabled subproject caching for the current build target (see *Configuring build extras* on page 10-20 for more information).

The file types you select are added to the search file list. See *Using file sets* on page 7-14 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. Select the **In Symbolics** tab in the **Find in Files** window (Figure 7-6).

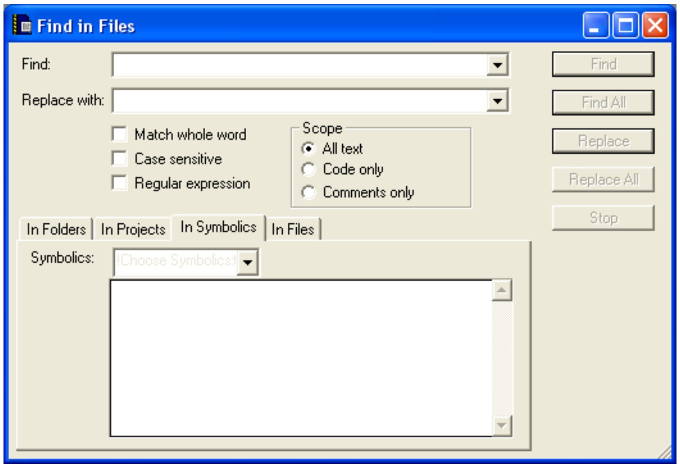


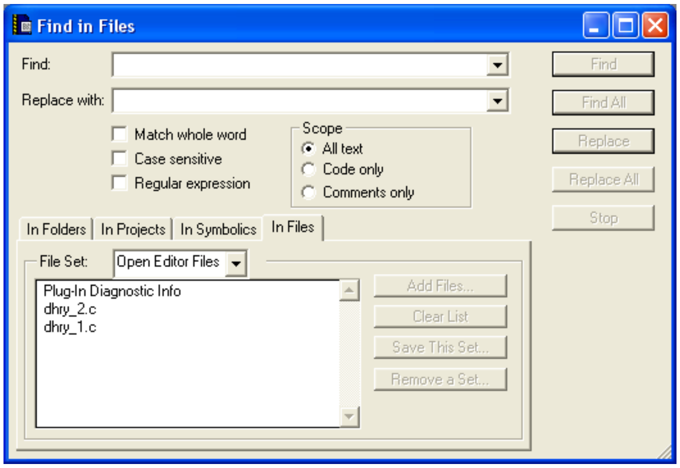
Figure 7-6 The In Symbols tab in the Find in Files Window

2. Select the files containing symbolics information you want to search in the **Symbolics** drop-down list. As you select files, they are added to the list at the bottom of the window. To remove a file from the list, select it and press the Delete button. 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.

Searching through multiple files by file

To search multiple files by file sets:

1. Select the **In Files** tab in the Find in Files window (Figure 7-7 on page 7-14).



Plug-In Diagnostic Info
dhry_2.c
dhry_1.c

Figure 7-7 The In Files tab in the Find in Files Window

2. Select the file set you want to search in the **File Set** drop-down list.
   - 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* 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.

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.

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 list.
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 on page 7-15).

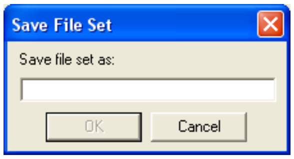


Figure 7-8 The Save File Set dialog

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 list.
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.

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).

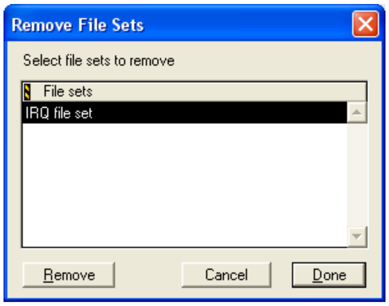


Figure 7-9 Remove File Sets dialog box

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 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 list.
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 list.
2. Click the **Clear List** button.

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 **Regexp** option is selected.

This section gives a brief introduction to regular expressions. For a comprehensive book on using regular expressions, refer to *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. See *Using regular expressions* on page 7-18 for more information.

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.

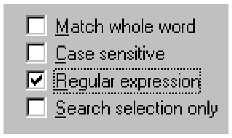
Table 7-1 Regular expression metacharacters (continued)

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).



☐ Match whole word
☐ Case sensitive
☑ Regular expression
☐ Search selection only

Figure 7-10 Regexp checkbox

3. Enter the search and replace strings. Your search and replace strings are treated as regular expressions.

The following examples show how to use regular expressions in search and replace operations.

Matching simple expressions

Most characters in a regular expression match themselves. The exceptions are the regular expression metacharacters listed in Table 7-1 on page 7-17. For example, the regular expression `a` matches all occurrences of the letter a in the search text.

To match a metacharacter literally, precede the metacharacter with a backslash. For example, to find every occurrence of a dollar sign ($), type `\$` in the Find text field. The backslash instructs the CodeWarrior IDE to interpret the dollar sign as a literal character, rather than a special character. If you do not use the backslash, the search finds end of line characters, not `$` characters.

Matching any character

A period (.) matches any character except a newline character or a null character.

For example, the regular expression:

```
var.
```

matches any four-character sequence that begins with `var`, such as `var1`, and `var2`.

Matching repeating expressions

The following metacharacters enable you to match repeating occurrences of a regular expression in your search string:

- A regular expression followed by an asterisk (*) matches *zero* or more occurrences of that regular expression. If there is any choice, the editor chooses the longest, left-most matching string in a line.
- A regular expression followed by a plus sign (+) matches *one* or more occurrences of that regular expression. If there is any choice, the editor chooses the longest, left-most matching string in a line.
- A regular expression followed by a question mark (?) matches *zero* or *one* occurrences of that regular expression. If there is any choice, the editor chooses the left-most matching string in a line.

Table 7-2 shows some simple examples.

Table 7-2 Using repetition operators

Regular expression	Matches
`s*ion`	Zero or more occurrences of the character `s` immediately preceding the characters `ion`. This regular expression matches with `ion` in `information` and `sections`, and with `ssion` in `expressions`.
`s+ion`	One or more occurrences of the character `s` immediately preceding the characters `ion`. This regular expression matches the `ssion` in `expressions`.
`s?ion`	Zero or one occurrences of the character `s` immediately preceding the characters `ion`. This regular expression matches with the `sion` in `expressions`, and with `ion` in `information` and `sections`.
`0\.?`	The number zero, followed by a period. The backslash tells the CodeWarrior IDE to treat the period as a literal character, and the `?` operator acts on the period character.

The asterisk, question mark, and plus metacharacters can operate on both single character regular expressions and grouped regular expressions. See *Grouping expressions* for details.

Grouping expressions

If an expression is enclosed in parentheses (`()`), it is treated as a single unit and repetition operators, such as the asterisk (`*`) or plus sign (`+`) are applied to the whole expression.

For example, to find strings that match `is`, you can type `is` in the Find text field. However, you can also use `( i)s` as a regular expression. This regular expression instructs the CodeWarrior IDE to look for the letter `s`, preceded by both a space and the letter `i`. Whereas `is` matches the `is` within `This`, `this`, and `is`, `( i)s` will match only with `is`.

Matching any character in a list

A string of characters enclosed in square brackets (`[]`) matches any one character in that string. For example, the regular expression:

```
[xyz]
```

matches any of the characters x, y, or z.

To match any character that is not in the string enclosed within the square brackets, precede the enclosed expression with a caret (^). For example, the regular expression:

```
[^abc]
```

matches every character in the search text other than a, b, and c.

To specify a range of consecutive ASCII characters, use a minus sign (-) within square brackets. For example, the regular expression:

```
[0-9]
```

is the same as:

```
[0123456789]
```

The following points apply to characters within the square brackets:

- If a minus sign is the first or last character within the square brackets, it is treated as a literal character. For example, the regular expression:

  ```
  [-bc]
  ```

  matches any one of the -, b, and c characters.

- A right square bracket immediately following a left square bracket does not terminate the string. It is considered to be one of the characters to match. For example, the regular expression:

  ```
  []0-9]
  ```

  matches the right square bracket and any digit.

- Metacharacters, such as backslash (\), asterisk (*), or plus sign (+), immediately following the opening square bracket are treated as literal characters. For example, the regular expression:

  ```
  [.]
  ```

  matches the period character.

You can use square brackets to group regular expressions in the same way as parentheses. The text string in the square brackets is treated as a single regular expression. For example, the regular expression:

```
[bsl]ag
```

matches any of `bag`, `sag`, or `lag`.

The regular expression:

```
[aeiou][0-9]
```

matches any lowercase vowel followed by a number, such as `a1`.

Matching the beginning or end of a line

You can specify that a regular expression matches the beginning or end of the line:

- If a caret (`^`) is at the beginning of the entire regular expression, it matches the beginning of a line. For example, the regular expression:

  ```
  ^([ \t]*cout)
  ```

  matches any occurrence of `cout` at the start of a line. The `[ \t]*` in the regular expression specifies that zero or more spaces or tabs can precede `cout`.

- If a dollar sign (`$`) is at the end of the entire regular expression, it matches the end of a line. For example, `this$` matches any occurrence of the string `this` at the end of a line.

- if an entire regular expression is enclosed by a caret and dollar sign (`^like this$`), it matches an entire line.

Using the find string in the replace string

You can specify the text found by a regular expression in the replace string by using an ampersand (`&`) in the replace regular expression. For example, if the find expression is `var[0-9]` and the replace string is `my_&`, the editor matches the find expression with strings such as `var1` and `var2` in the search text, and replaces `var1` with `my_var1` and `var2` with `my_var2`.

Use `\&` to specify a literal ampersand in the replace string. An ampersand has no special meaning in the find string.

Using subexpressions in the replace string

You can specify subexpressions of a regular expression in a find string, and use the subexpressions in the replace string. You can specify up to nine subexpressions for each find string. Each subexpression must be enclosed within parentheses.

To use a subexpression in the replace string, type `\`*`n`*, where *`n`* is a digit that specifies which subexpression to recall. Subexpressions are counted from the left side of the find string to determine the value of *`n`*.

For example, to change `#define` declarations of the form:

```
#define var1 10
```

into **`const`** declarations:

1. Select **Find…** from the **Search** menu and ensure that the **Regexp** option is selected.

2. Enter the following regular expression in the Find text field:

   ```
   \#define[ \t]+(.+)[ \t]+([0-9]+);
   ```

   This regular expression matches string patterns of the following form:

   `#define`, followed by `one or more spaces or tabs`, followed by `one or more characters`, followed by `one or more spaces or tabs`, followed by `one or more digits`, followed by a `semicolon`.

   Starting from the left side of the find regular expression, the first subexpression is `(.+)`, and the second subexpression is `([0-9]+)`.

3. Enter the following regular expression in the Replace text box:

   ```
   const int \1 = \2;
   ```

   The `\1` specifies the text found by the first subexpression. The `\2` specifies the text found by the second subexpression. The two subexpressions recall the variable name and its value from the original `#define` declaration.

4. Click **Replace** when the string is found. The replace string changes the `#define` declaration into a `const` declaration by using references to the two subexpressions. For example:

   ```
   #define var1 10;
   ```

   is changed to:

   ```
   const int var1 = 10;
   ```

Chapter 8
Working with the Browser

This chapter describes the CodeWarrior browser. The CodeWarrior browser provides you with a user interface to access a database of all the symbols in your code quickly and easily. The symbol database is generated by the language parser when the browser is activated and you build your project. The CodeWarrior browser works with both procedural and object-oriented code.

This chapter contains the following sections:

8.1 About working with the browser

The CodeWarrior 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. This is the default setting for all ARM stationery projects. See *Configuring build extras* on page 10-20.

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 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.

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. Figure 8-7 on page 8-17 shows an example of a contents view.

See *Viewing data by type with the Contents view* on page 8-17 for details of the Contents window interface.

Class browser view

You can use the Class browser view to examine your code from a class-based perspective. Figure 8-2 on page 8-9 shows an example.

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 view* on page 8-8 for 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. Figure 8-8 on page 8-18 shows an example.

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 details on the Hierarchy view interface.

The 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 8-21 for more information.

In addition, the 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 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

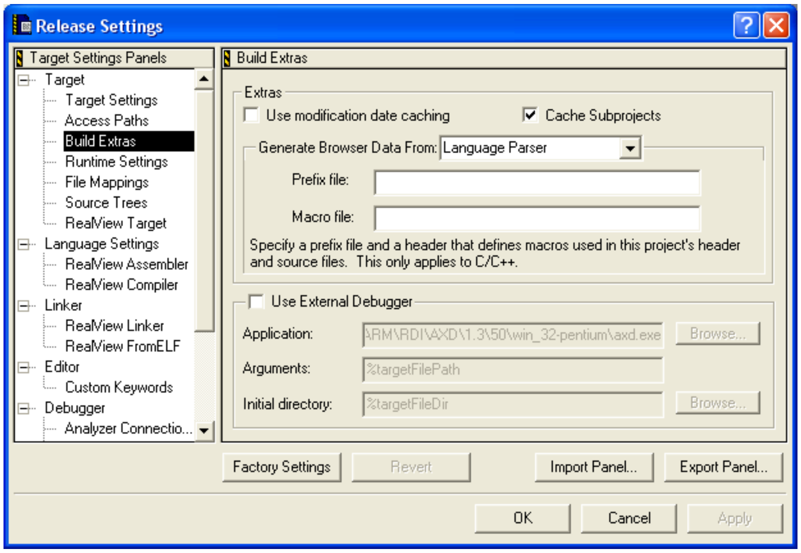
The browser-related menu commands in the **View** menu (**Browser Contents window**, **Class Hierarchy window**, and **Class Browser**) display wide-focus views. After you have selected a wide view, you can use a context-sensitive menu to focus on a particular class.

8.2 Enabling browser data production

You must enable the production of browser data for each target in your project; browse data is not shared between targets. 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 Settings panel for your project.

2. Click **Build Extras** in the Target Settings Panels list. The CodeWarrior IDE displays the Build Extras panel (Figure 8-1).



Build Extras

Extras

☐ Use modification date caching ☑ Cache Subprojects

Generate Browser Data From: Language Parser

Prefix file:

Macro file:

Specify a prefix file and a header that defines macros used in this project's header and source files. This only applies to C/C++.

☐ Use External Debugger

Application: ARM\RDI\AXD\1.3\50\win_32-pentium\axd.exe

Arguments: %targetFilePath

Initial directory: %targetFileDir

Figure 8-1 Build extras settings panel

3. Choose **Language Parser** from the drop-down list box 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 panel.

See *Configuring browser options* on page 8-6 for more details on browser settings and options.

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 subprojects*.

——— **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 9-35 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 subprojects

To include browser information from subprojects of the current build target you must enable subproject caching.

——— **Note** ———

This option is selected by default.

———————

To enable subproject caching:

1. Display the Target Settings panel for the project you want to configure (see *Displaying Target Settings panels* on page 10-4) for more information.
2. Click **Build Extras** in the Target Settings Panels list. The CodeWarrior IDE displays the Build Extras panel.

3. Select the **Cache Subprojects** checkbox. This option:
   - improves multiproject updating and linking
   - enables the Class browser to include browser information from target subprojects.

   However, this option also increases the amount of memory required by the CodeWarrior IDE.

   See *Configuring build extras* on page 10-20 for more information on the Build Extras panel.
4. Click **Apply** to apply your changes.

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 view*
- *Viewing data by type with the Contents view* 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 view

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

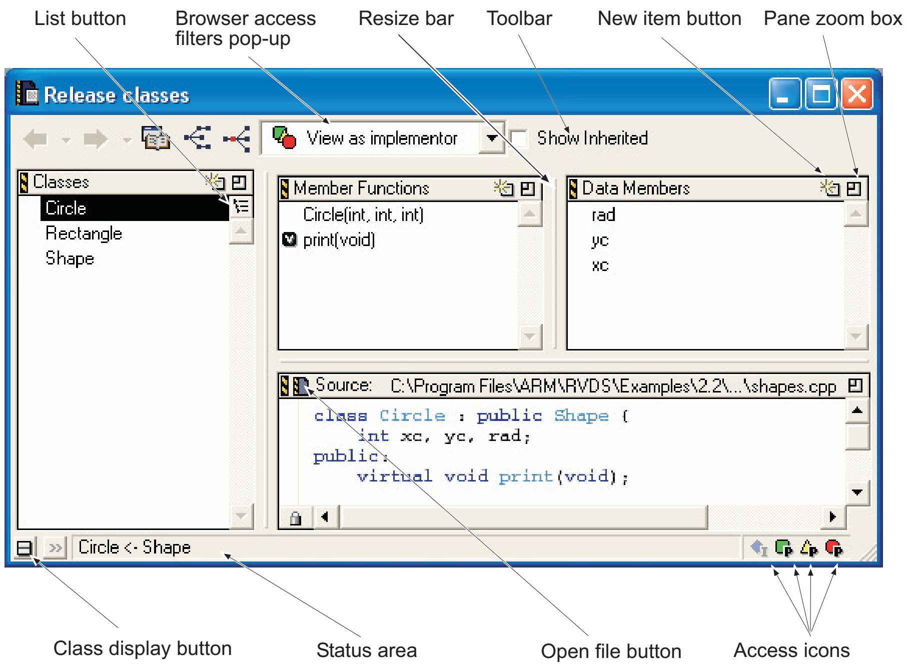
- You must use the File Mappings configuration panel to map the ARM RealView compiler to process all C++ source and header files containing classes. By default, header files are mapped to the ARM RealView compiler.

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 (Figure 8-2 on page 8-9).



```
class Circle : public Shape {
    int xc, yc, rad;
public:
    virtual void print(void);
```

Figure 8-2 A Class browser view

The main components of the Class browser window are:

Browser toolbar

The browser toolbar provides access to a number of CodeWarrior IDE commands, including Go Forward and Go Back navigation buttons, and buttons to open hierarchy view windows. See *Finding declarations, definitions, overrides, and multiple implementations* on page 8-24 for more information.

Browser Access Filters drop-down list

Use this 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.

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** checkbox 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.

———

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** checkbox 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.

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, refer to 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 (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.

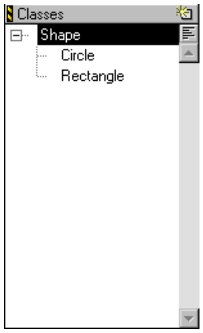


Figure 8-3 The Classes pane

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.

Caution

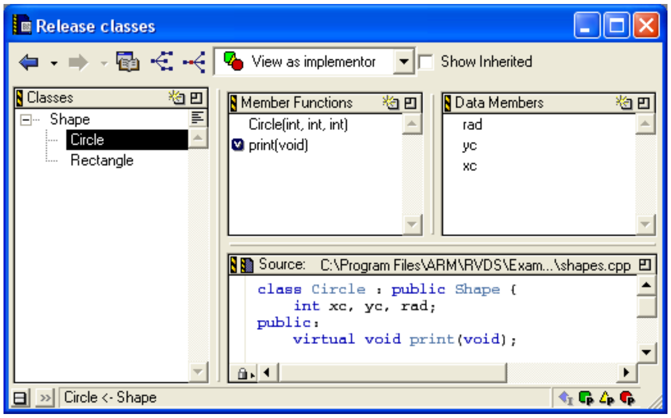
If the Source pane is active and you press the Tab key, a tab is entered into your source code.

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 panels 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.



```
class Circle : public Shape {
    int xc, yc, rad;
public:
    virtual void print(void);
```

Figure 8-4 Member functions, data members, and declaration for class circle

4. 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
- *Opening another view from the Class browser view* on page 8-16

- *Using the Browser context menu* on page 8-21
- *Finding declarations, definitions, overrides, and multiple implementations* on page 8-24
- *Editing code in the browser* on page 8-28.

Filtering members by access type

You can use the **Access Filters** drop-down list to filter the display of member functions and data members in the Class browser view. The drop-down list 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 list in the Class browser toolbar (Figure 8-5).

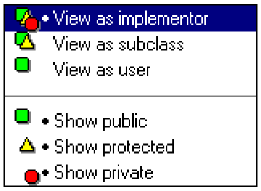


Figure 8-5 Access filters drop-down list

The drop-down list displays a list of access types. A bullet is displayed in the menu next to each access type currently selected.

4. Select the access type you want from the drop-down list:

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.

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 (see Figure 8-6).

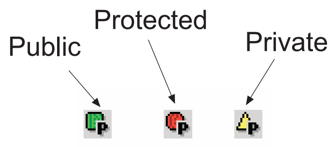


Figure 8-6 Browser access filter icons

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 8-21 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.
- The 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.
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 view

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.

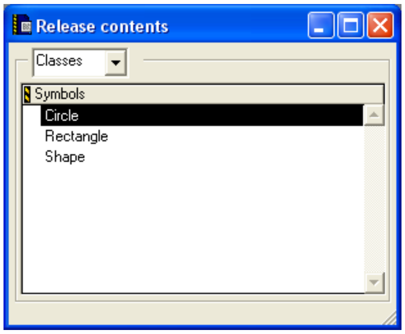


Figure 8-7 A Contents window

3. Select the category of data you want to view from the **Category** drop-down list 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.

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 8-21 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 on 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 8-21 for more information.

   The CodeWarrior IDE displays a hierarchy window for the current build target (Figure 8-8).

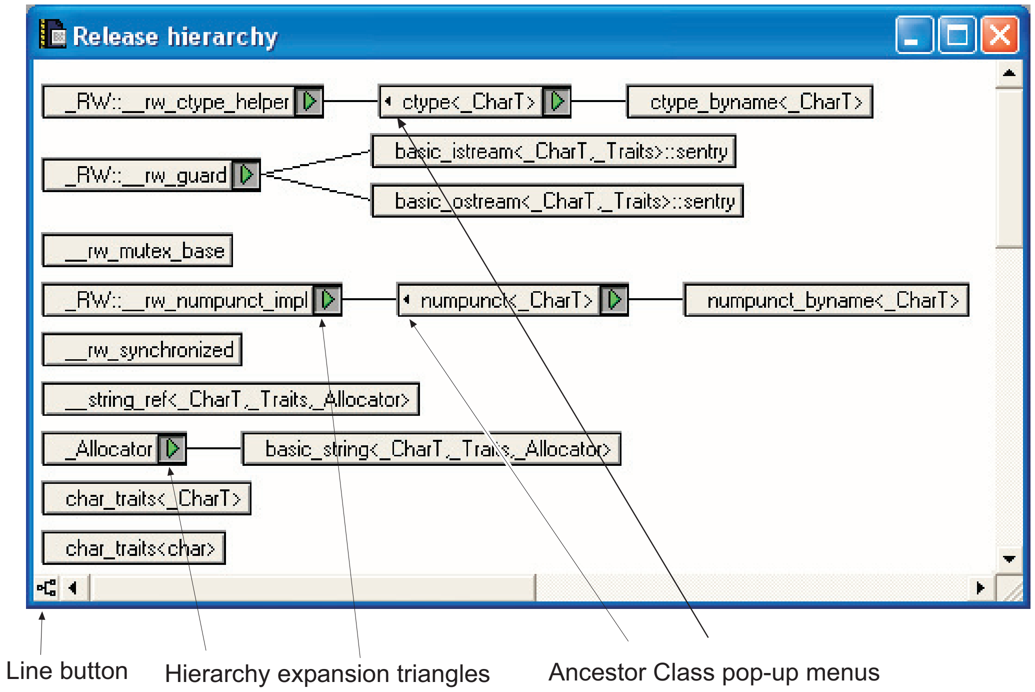


Figure 8-8 The Multi-class hierarchy window

2. Select **Class Hierarchy Window** from the **View** menu. Alternatively, click on the **Show hierarchy** button in the Class browser window. The CodeWarrior IDE displays a hierarchy window for the current build target (Figure 8-8 on page 8-18)

   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. 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 called 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.

   Ancestor Class drop-down list

   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 list. Figure 8-9 shows an example.

   Select the ancestor class you want from the drop-down list to jump to the hierarchy view for the ancestor class.

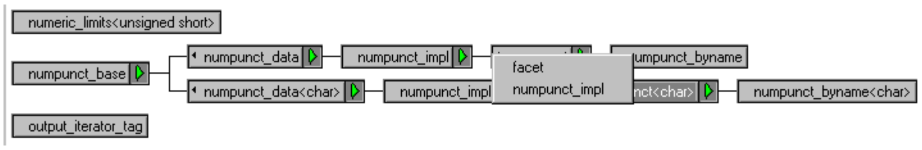


Figure 8-9 Ancestor drop-down list

3. Navigate to the class you want to view:
   - Use the arrow keys to change the selected class:
     — use the up and down key to move between siblings

    - use the left and right 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 view* 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 8-21 for more information.

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*
- *Finding declarations, definitions, overrides, and multiple implementations* on page 8-24
- *Using symbol name completion* on page 8-27
- *Editing code in the browser* on page 8-28.

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:

- Click the **Go Back** or **Go Forward** buttons in the browser toolbar.

- 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).

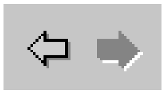


Figure 8-10 Go Back and Go Forward toolbar buttons

- Select **Go Back** or **Go Forward** from the **Search** menu.

——— **Note** ———

Go Back and **Go Forward** do not undo any actions you performed.

———————————

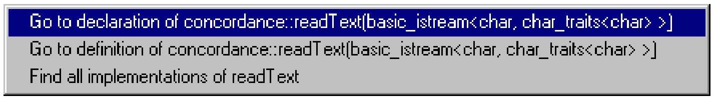
8.4.2 Using the Browser context menu

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 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.



Go to declaration of concordance::readText(basic_istream<char, char_traits<char> >)
Go to definition of concordance::readText(basic_istream<char, char_traits<char> >)
Find all implementations of readText

Figure 8-11 A Browser context menu for a function

For member functions, you can:

- View the function declaration. See *Viewing a class or member declaration* on page 8-24.
- 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 8-25.

Using the Browser context menu from an editor window

In the 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.

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 8-23 for more information.

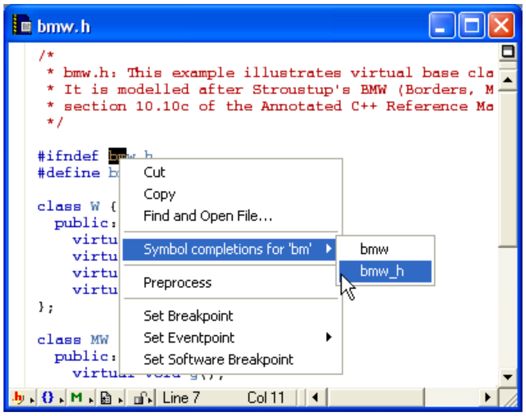
Note

- 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 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.



```
/*
 * bmw.h: This example illustrates virtual base cla
 * It is modelled after Stroustup's BMW (Borders, M
 * section 10.10c of the Annotated C++ Reference Ma
 */

#ifndef bmw_h
#define b

class W {
  public:
    virtu
    virtu
    virtu
    virtu
};

class MW
  public:
    virtual void g();
```

Figure 8-12 Using symbol name completion

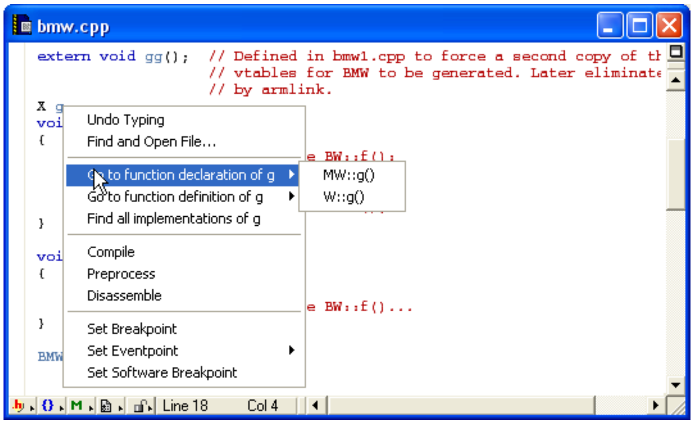
2. Select an item from the list to enter it into your text file. See *Using symbol name completion* on page 8-27 for other ways to type browser items.

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 **Include insert template commands** option is selected in the Browser Display configuration panel. See *Editor settings* on page 9-23 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 on page 8-24 shows an example.



```
extern void gg();  // Defined in bmw1.cpp to force a second copy of th
                   // vtables for BMW to be generated. Later eliminate
                   // by armlink.
```

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.

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* on page 8-25
- *Finding overrides and multiple implementations of a function* on page 8-25.

Viewing a class or member declaration

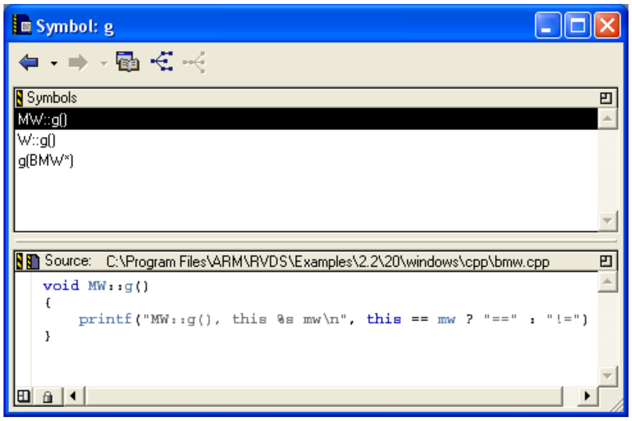
Use any of the following methods to display a class or member declaration:

- 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 the 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 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 shows an example for the symbol `g`.



MW::g()
W::g()
g(BMW*)

C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.cpp

```
void MW::g()
{
    printf("MW::g(), this %s mw\n", this == mw ? "==" : "!=")
}
```

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

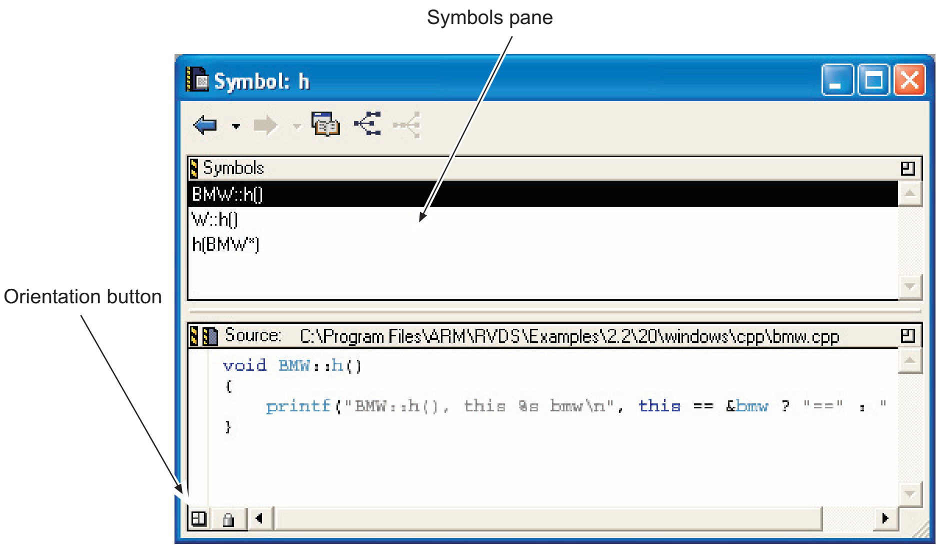
The Symbol window lists all implementations of any symbol that has multiple definitions. Typically, these symbols are multiple versions of overridden functions. However, the Symbol window works for any symbol that has multiple definitions in the browser database.

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 V. 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).

   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.


Symbols pane
Orientation button
Symbol: h
Symbols
BMW::h()
W::h()
h(BMW*)
Source: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.cpp

```
void BMW::h()
{
    printf("BMW::h(), this %s bmw\n", this == &bmw ? "==" : "
}
```

Figure 8-15 The 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 view* on page 8-8 for more information. The Symbol window has two items 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.

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 and Get previous symbol

Type Ctrl-. after using one of the **Find symbols** commands to search for the next symbol in the database that matches your search string.

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 8-21 for more details.

8.4.5 Editing code in the browser

Code displayed in a Source code pane is fully editable. You can use standard CodeWarrior 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.

8.5 Creating classes and members with browser wizards

When you open a Contents View, Browser View, or Hierarchy View 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*.

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 8-34.

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 8-36.

Note

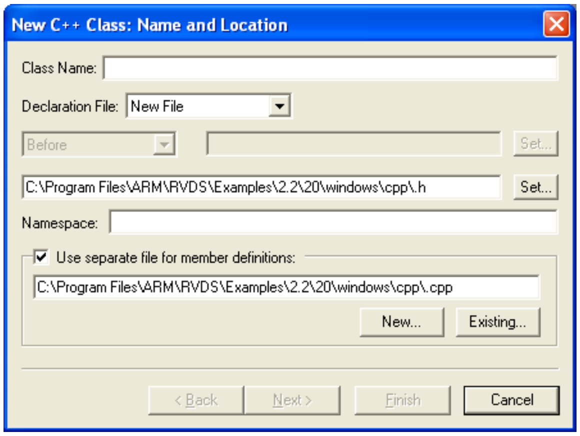
The **New Property…**, **New Method…**, **New Event Set…**, and **New Event…** menu items are not implemented by the CodeWarrior IDE for RVDS.

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.

To create a new class with the New Class wizard:

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 Name and Location page of the New C++ Class Wizard (Figure 8-16).



New C++ Class: Name and Location

Class Name:

Declaration File: New File

Before

C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\.h

Namespace:

☑ Use separate file for member definitions:

C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\.cpp

Figure 8-16 New C++ Class: 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 list to specify the type of declaration file. Depending on the option you choose, different fields become enabled below the **Declaration File** drop-down list. You can select either:

New File Select this option to create a new declaration file. Enter the pathname for the new file, or click **Set…** to use the standard file dialog to set a directory for the new file. By default the file is saved with the name `*classname*.h`.

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 list to place the new class **Before** or **After** the selected class declaration.

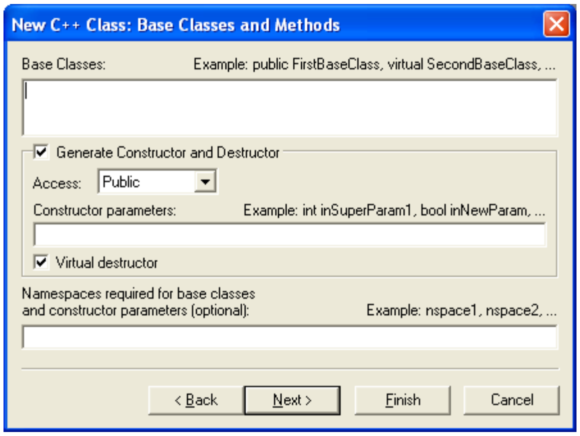
Namespace

Namespaces are not supported by the RealView compiler. Leave this field empty.

Use separate file for member definitions

Select this checkbox 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 checkbox, 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 Class and methods page is displayed (Figure 8-17).


New C++ Class: Base Classes and Methods

Base Classes: Example: public FirstBaseClass, virtual SecondBaseClass, ...

☑ Generate Constructor and Destructor

Access: Public

Constructor parameters: Example: int inSuperParam1, bool inNewParam, ...

☑ Virtual destructor

Namespaces required for base classes and constructor parameters (optional): Example: nspace1, nspace2, ...

Figure 8-17 New C++ Class: Base Class and Methods

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.

Generate Constructor and Destructor

Select this checkbox 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 list.

Constructor parameters

Enter a list of parameters for the constructor. Example parameters are listed above the field.

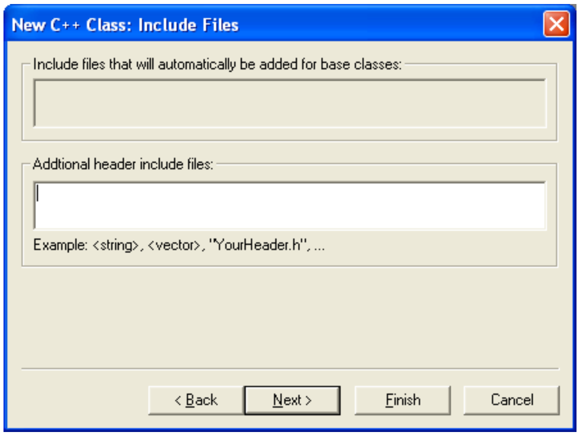
Virtual destructor

Select this checkbox 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 Include Files page of the New Class wizard is displayed (Figure 8-18).


New C++ Class: Include Files

Include files that will automatically be added for base classes:

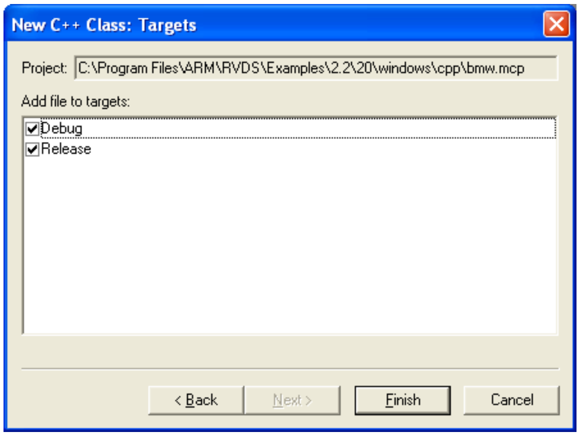
Addtional header include files:

Example: <string>, <vector>, "YourHeader.h", ...

Figure 8-18 New C++ Class: Include Files

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 Targets page is displayed (Figure 8-19 on page 8-33).



Project: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.mcp

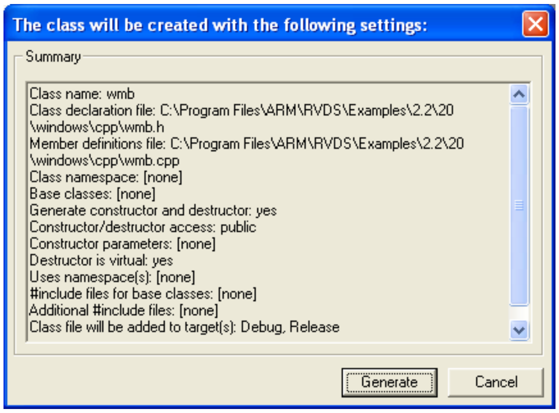
Add file to targets:

☑ Debug
☑ Release

Figure 8-19 New C++ Class: Targets

9. Select the checkbox 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).



```
Class name: wmb
Class declaration file: C:\Program Files\ARM\RVDS\Examples\2.2\20
\windows\cpp\wmb.h
Member definitions file: C:\Program Files\ARM\RVDS\Examples\2.2\20
\windows\cpp\wmb.cpp
Class namespace: [none]
Base classes: [none]
Generate constructor and destructor: yes
Constructor/destructor access: public
Constructor parameters: [none]
Destructor is virtual: yes
Uses namespace(s): [none]
#include files for base classes: [none]
Additional #include files: [none]
Class file will be added to target(s): Debug, Release
```

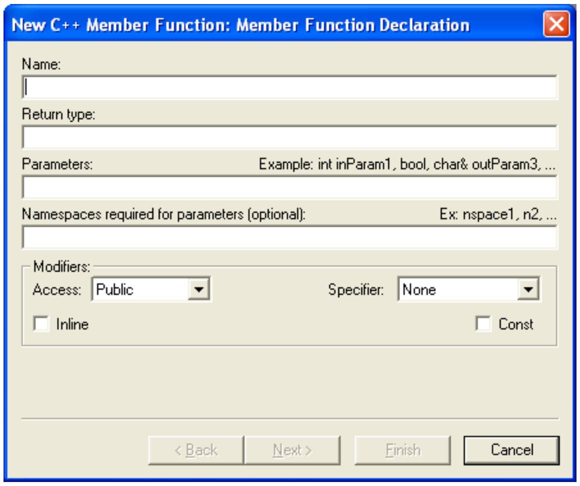
Figure 8-20 New class summary

11. Click **Generate** to create the new class.

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. To create a new member function with the Member Function wizard:

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 Declaration panel of the New Member Function Wizard (Figure 8-21).


New C++ Member Function: Member Function Declaration

Name:

Return type:

Parameters: Example: int inParam1, bool, char& outParam3, ...

Namespaces required for parameters (optional): Ex: nspace1, n2, ...

Modifiers:
Access: Public
Specifier: None
Inline
Const

Figure 8-21 New Member Function: Member Function Declaration

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.

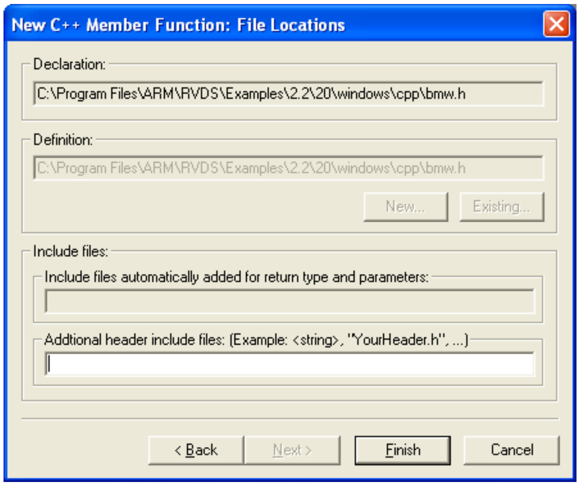
   Namespaces required for parameters (optional)
   Enter any namespace required for the parameters you are adding.

Modifiers

Modify the function declaration, as required:

- Use the **Access** drop-down list to specify whether the new member function is public, protected, or private.
- Use the **Specifier** drop-down list if you want to declare the new member function as a virtual, pure virtual, or static function.
- Select the **Inline** or **Const** check boxes to declare the function `inline` or `const`. If you select the Inline checkbox, 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 File Locations page is displayed (Figure 8-22).


New C++ Member Function: File Locations
Declaration:
Definition:
New...
Existing...
Include files:
Include files automatically added for return type and parameters:
Addtional header include files: (Example: <string>, "YourHeader.h", ...)
< Back
Next >
Finish
Cancel

Declaration: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.h

Definition: C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.h

Figure 8-22 New Member Function: File Locations

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 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.

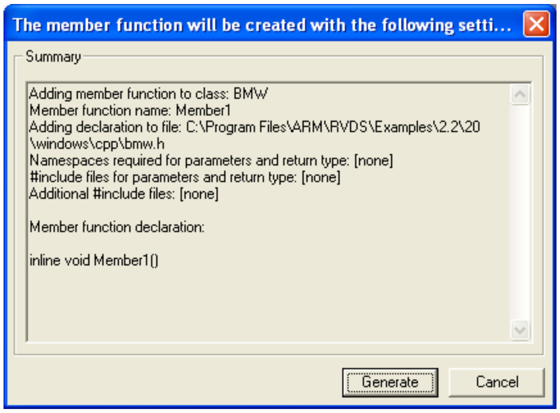
Include files automatically added…

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).



The member function will be created with the following setti...
Summary
Generate
Cancel

```
Adding member function to class: BMW
Member function name: Member1
Adding declaration to file: C:\Program Files\ARM\RVDS\Examples\2.2\20
\windows\cpp\bmw.h
Namespaces required for parameters and return type: [none]
#include files for parameters and return type: [none]
Additional #include files: [none]

Member function declaration:

inline void Member1()
```

Figure 8-23 New member function declaration summary

8. 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 checkbox is selected.

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. To create a new data member for a 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 Data Member Declaration page of the New Data Member Wizard (Figure 8-24).

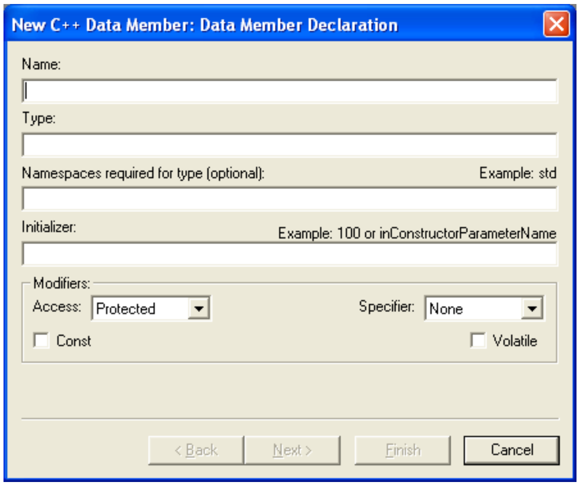

New C++ Data Member: Data Member Declaration
Name:
Type:
Namespaces required for type (optional):
Example: std
Initializer:
Example: 100 or inConstructorParameterName
Modifiers:
Access:
Protected
Specifier:
None
Const
Volatile
< Back
Next >
Finish
Cancel

Figure 8-24 New Data Member wizard: Data Member Declaration

4. Declare the new data member:

Name Enter a name for the data member.

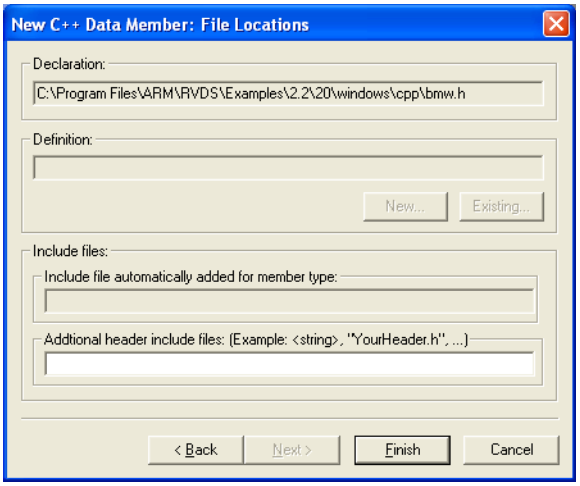
Type Enter the data member type.

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.

Modifiers
Use the **Access** and **Specifier** drop-down lists 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** checkboxes as desired 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 File Locations page is displayed (Figure 8-25 on page 8-38).


New C++ Data Member: File Locations

Declaration:
C:\Program Files\ARM\RVDS\Examples\2.2\20\windows\cpp\bmw.h

Definition:

Include files:
Include file automatically added for member type:

Addtional header include files: (Example: <string>, "YourHeader.h", ...)

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.

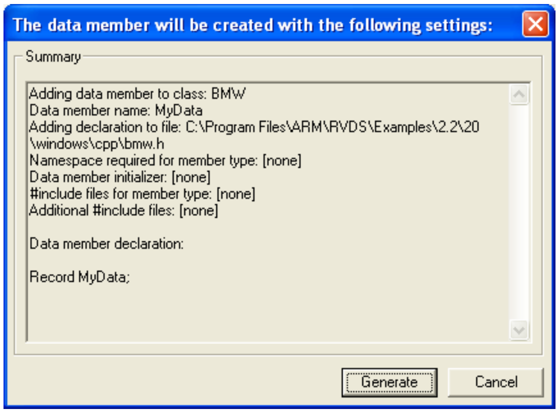
   Include file automatically added for member type

   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.

7. Click **Finish**. The CodeWarrior IDE displays a summary of the information you have entered for the new data member (Figure 8-26 on page 8-39).


The data member will be created with the following settings:
Summary

```
Adding data member to class: BMW
Data member name: MyData
Adding declaration to file: C:\Program Files\ARM\RVDS\Examples\2.2\20
\windows\cpp\bmw.h
Namespace required for member type: [none]
Data member initializer: [none]
#include files for member type: [none]
Additional #include files: [none]

Data member declaration:

Record MyData;
```

Figure 8-26 New data member summary

8. Click **Generate** to generate source for the new data member. The CodeWarrior IDE adds a data member declaration to the selected class.

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 keybindings for commands. It contains the following sections:

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 editor. See *Choosing editor preferences* on page 9-20 for more information.

Customizing commands and keybindings

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 9-38 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.

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.

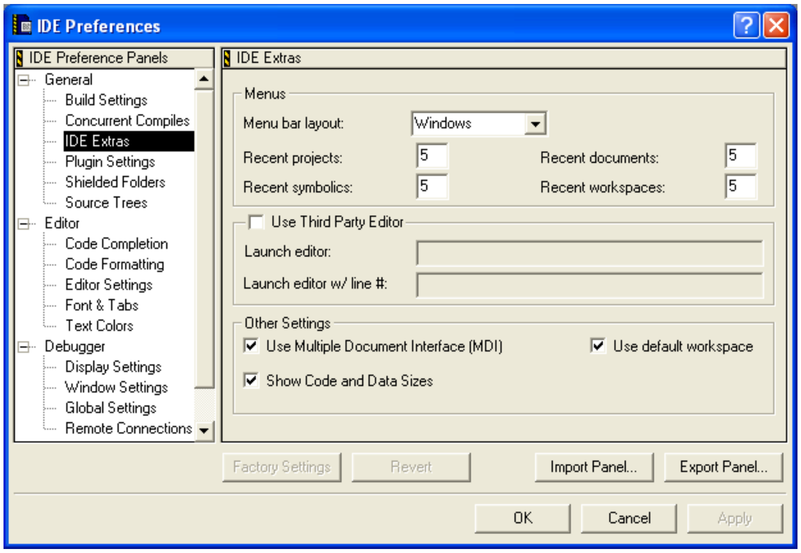
Opening the IDE Preferences panel

To open the IDE Preferences panels 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.

   Note

   The Debugger preferences panels are not used by the CodeWarrior IDE for RVDS.



Menu bar layout: Windows
Recent projects: 5
Recent documents: 5
Recent symbolics: 5
Recent workspaces: 5
☐ Use Third Party Editor
Launch editor:
Launch editor w/ line #:
Other Settings
☑ Use Multiple Document Interface (MDI)
☑ Use default workspace
☑ Show Code and Data Sizes

Figure 9-1 The IDE Preferences panel

2. Select the panel you want to configure from the list. You can use the arrow keys or click the name of the panel.

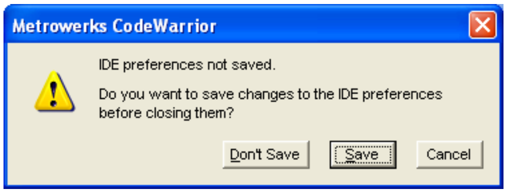
   Each panel contains related options that you can set. The options you select apply to all CodeWarrior IDE projects.

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

If you make changes in the IDE Preferences window and attempt to close it, the CodeWarrior IDE displays a Preferences Confirmation dialog box like that shown in Figure 9-2.


Metrowerks CodeWarrior

IDE preferences not saved.

Do you want to save changes to the IDE preferences before closing them?

Figure 9-2 Preferences Confirmation dialog box

Click either:

- **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

In addition, you can use the dialog box buttons in the IDE Preferences window to apply or discard your changes. The dialog buttons are:

OK	Click this button to save any changes that you have made and close the panel.
Cancel	Click this button to close the panel 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 panel and then decide not to use them for the current panel.

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.

——— Note ———

The **Import Panel** and **Export Panel** features are useful if you want to copy *all* the settings for a panel to the same panel in another target. If you only want to copy some panel settings - for example, the setttings for one tab in a panel - you must edit the panel manually.

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.

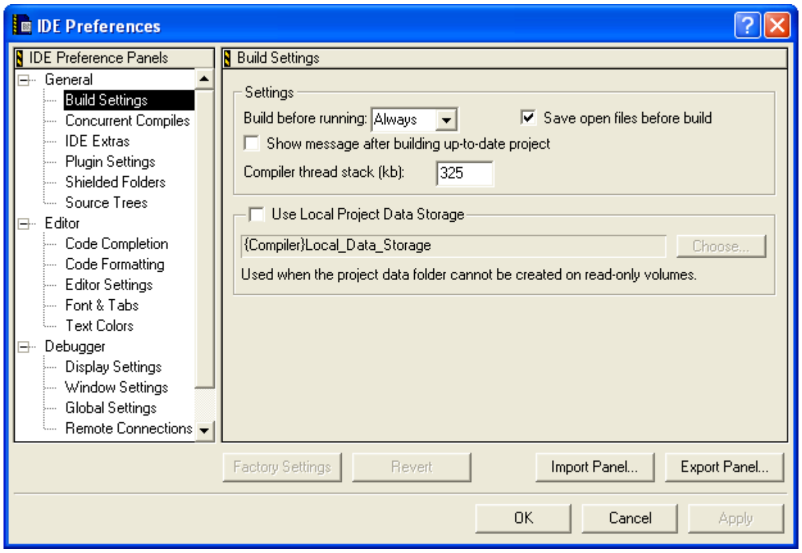
This section describes:

- *Configuring build settings*
- *Configuring concurrent compiles* on page 9-8
- *Configuring IDE extras* on page 9-9
- *Configuring plug-in settings* on page 9-12
- *Configuring shielded folders* on page 9-14
- *Configuring global source trees* on page 9-16.

9.3.1 Configuring build settings

The Build Settings panel enables you to customize a number of project build settings. To open the Build Settings 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 panel (Figure 9-3).



Build Settings

Settings

Build before running: Always

[✓] Save open files before build

[ ] Show message after building up-to-date project

Compiler thread stack (kb): 325

[ ] Use Local Project Data Storage

{Compiler}Local_Data_Storage

Used when the project data folder cannot be created on read-only volumes.

Figure 9-3 Build Settings preference panel

2. Change the following options, as required:

   Build before running

   Use this drop-down list 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 choose:

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. The up-to-date project is not built.

   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 used by the CodeWarrior IDE for RVDS.

   Use Local Project Data Storage

   After loading a project file, the IDE creates or updates an associated project data folder. The IDE stores intermediate project data in this folder. After you build or close the project, the IDE uses the information in the project data folder to update the project file.

   By default, the IDE places the project data folder within the same folder as the project file. However, the IDE cannot create or update a project data folder in a location that grants read-only privileges. To specify a different location for the project data folder, enable the Use Local Project Data Storage checkbox, then click **Choose** to select the path to the location you wish to use. For more information on the types of paths available to you, see *Configuring global source trees* on page 9-16.

3. Click **Apply** to apply your changes.

9.3.2 Configuring concurrent compiles

The Concurrent Compiles panel provides you with an option to enable concurrent compilations for your project. When this feature is enabled, CodeWarrior improves its use of available processor capacity by spawning multiple compile processes to compile code more efficiently. When concurrent compilations are enabled, compilation is improved by:

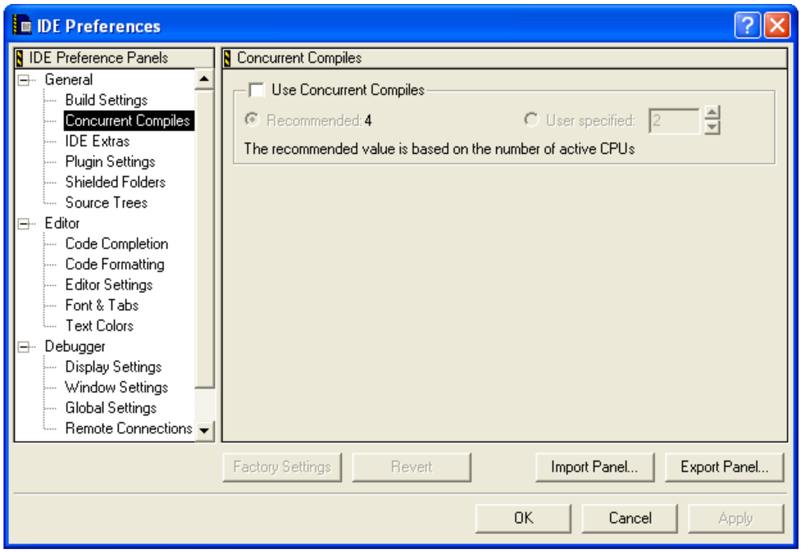
- Optimizing resource use
- Using overlapped input/output

Note

You must have enough ARM licenses available for the number of concurrent compilations that you wish to use.

To open the Concurrent Compiles 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 panel (Figure 9-5 on page 9-9).



Use Concurrent Compiles [ ]

Recommended: 4 [●] User specified: 2 [○]

The recommended value is based on the number of active CPUs

Figure 9-4 Concurrent Compiles preference panel

2. Click **Use Concurrent Compiles** to enable concurrent compiles.

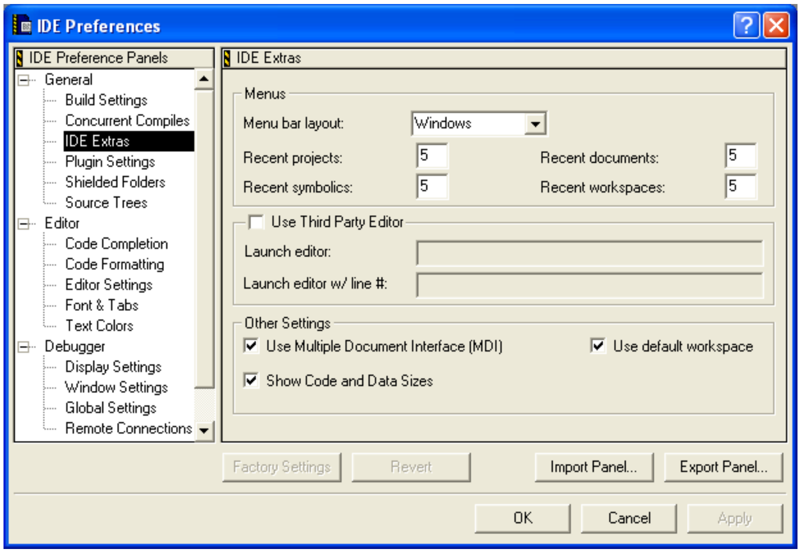
3. Click on **Recommended** or **User specified** to specify the number of compilation processes which can be run simultaneously.

4. Click **Apply** to apply your changes.

9.3.3 Configuring IDE extras

The IDE Extras 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 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 panel (Figure 9-5).



Menu bar layout: Windows

Recent projects: 5 Recent documents: 5

Recent symbolics: 5 Recent workspaces: 5

☐ Use Third Party Editor

Launch editor:

Launch editor w/ line #:

☑ Use Multiple Document Interface (MDI) ☑ Use default workspace

☑ Show Code and Data Sizes

Figure 9-5 IDE Extras preference panel

There are three groups of options. For details of how to change the options see:

- *Configuring the Menus submenu*
- *Using a third-party text editor* on page 9-10
- *Other settings* on page 9-11.

Configuring the Menus submenu

Use the **Menu bar layout** drop-down list to choose 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 IDE menus according to Microsoft® Windows® user-interface standards. Appendix D *CodeWarrior IDE Reference* shows the organization of the menu commands under each menu bar layout.

To choose the default menu layout:

1. Open the IDE Extras panel (see *Configuring IDE extras* on page 9-9).
2. Select either **Windows** or **Macintosh** from the **Menu bar layout** drop-down list.

Note

The CodeWarrior IDE for the RealView Development Suite does not use the Macintosh layout, and only the Windows layout is documented in this User Guide.

You can configure how many projects and documents are displayed in the **File** → **Open Recent** submenu. To set the number of project and documents displayed:

1. Open the IDE Extras 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 the CodeWarrior IDE to display in the **File** → **Open Recent** submenu.

   Recent documents
   Enter the maximum number of files you want the CodeWarrior IDE to display in the **File** → **Open Recent** submenu.

   Recent symbolics
   Enter the maximum number of symbolics you want the CodeWarrior IDE to display in the **File** → **Open Recent** submenu.

   Recent workspace
   Enter the maximum number of workspaces you want the CodeWarrior IDE to display in the **File** → **Open Recent** submenu.

   Note

   CodeWarrior IDE for the RealView Development Suite does not use symbolic files.

   ---

3. Click **Apply** to apply your changes.

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** panel (see *Configuring IDE extras* on page 9-9).

2. Select the **Use Third Party Editor** checkbox. When this checkbox is selected, the CodeWarrior IDE uses the third-party text editor you specify to open text files.

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 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 pathname 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 10-20 for more information.

4. Click **Apply** to apply your changes.

Other settings

The Other Settings group box has three options that enables 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 the MDI interface:

1. Open the IDE Extras panel (see *Configuring IDE extras* on page 9-9).

2. Select the **Use Multiple Document Interface** checkbox to use the Windows *Multiple Document Interface* (MDI).

   Deselect the checkbox 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 3-21.

To use the default workspace:

1. Open the IDE Extras panel (see *Configuring IDE extras* on page 9-9.

2. Select the **Use default workspace** checkbox. When this checkbox is selected the state of all open IDE windows will be retained when you exit the IDE. Deselect this option to open the IDE with no windows visible.

For more information about workspaces, see *Working with project workspaces* on page 3-25.

To show code and data sizes:

1. Open the IDE Extras panel (see *Configuring IDE extras* on page 9-9.

2. Select the **Show Code and Data Sizes** checkbox to turn off the display of code and data sizes for project files in the project window.

For more information about displaying information in the Project window, see *Project window Files view* on page 3-7.

9.3.4 Configuring plug-in settings

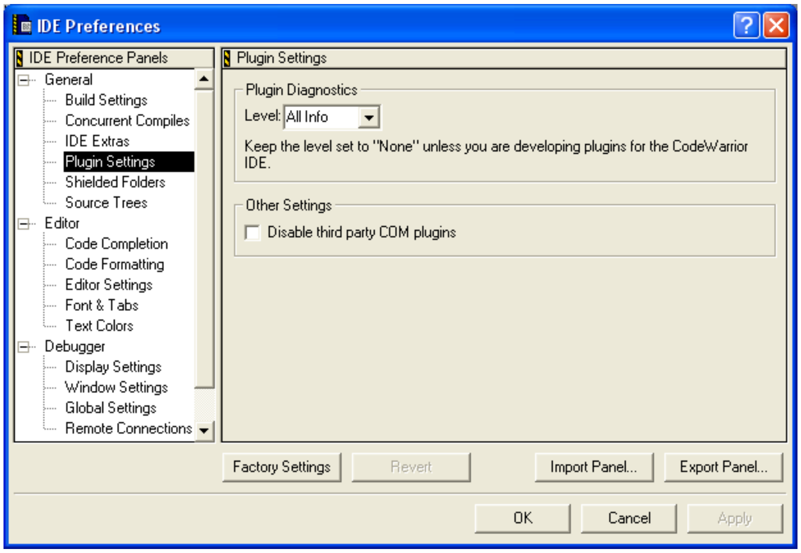
Use the plug-in Settings panel to specify how much plug-in diagnostic information the CodeWarrior IDE provides. Plug-in diagnostic information is useful if you are using the CodeWarrior IDE to develop plug-ins for the CodeWarrior IDE. Use this panel if you have problems getting your plug-in to function properly, or if you want more information about the properties of installed plug-ins.

Note

You cannot develop CodeWarrior plug-ins with RealView Developer Suite tools. However, if you develop a CodeWarrior 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 Plugin Settings in the IDE Preference Panels list. The CodeWarrior IDE displays the Plugin Settings panel (Figure 9-6).



Plugin Settings

Plugin Diagnostics

Level: All Info

Keep the level set to "None" unless you are developing plugins for the CodeWarrior IDE.

Other Settings

☐ Disable third party COM plugins

Figure 9-6 Plugin settings panel

2. Select the level of plug-in diagnostics you want. You can specify three levels of plug-in diagnostics:

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.

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 COMplugins** if you want to disable all third-party *Common Object Model* (COM) plug-ins for the CodeWarrior IDE. This can be helpful for troubleshooting purposes.

   Note

   If you experience problems while 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 panel enables you to configure the CodeWarrior IDE to ignore specified folders during certain operations. This option is useful if you want to avoid collisions between example files or version control data, for example. You specify regular expressions that match the names of the folders that you want to ignore. The IDE treats matching folders as shielded folders and ignores their contents. For more information about regular expressions, see *Using regular expressions* on page 7-18.

The Shielded Folders list at the top of the panel shows the current list of regular expressions. The IDE uses this list to determine whether to shield folders from the current project. The following regular expressions appear in the Shielded Folders list by default:

- The `\(.*\)` regular expression matches folders with names that begin and end with parentheses, such as `(Project Stationery)`.
- The `CVS` regular expression matches folders named CVS. With this regular expression, the IDE skips data files used by the CVS version control system
- The `.*[_]Data` regular expression matches the names of folders used by the IDE to store target data information, such as `MyProject_Data`.

The Shielded Folders list includes two columns. These columns represent different types of operations for which you can shield folders:

A bullet in the Project operations column indicates that the IDE ignores matching folders for project operations. Such operations include dragging a folder into the Project window, building a project, and searching access paths for source or include files after choosing the Open command.

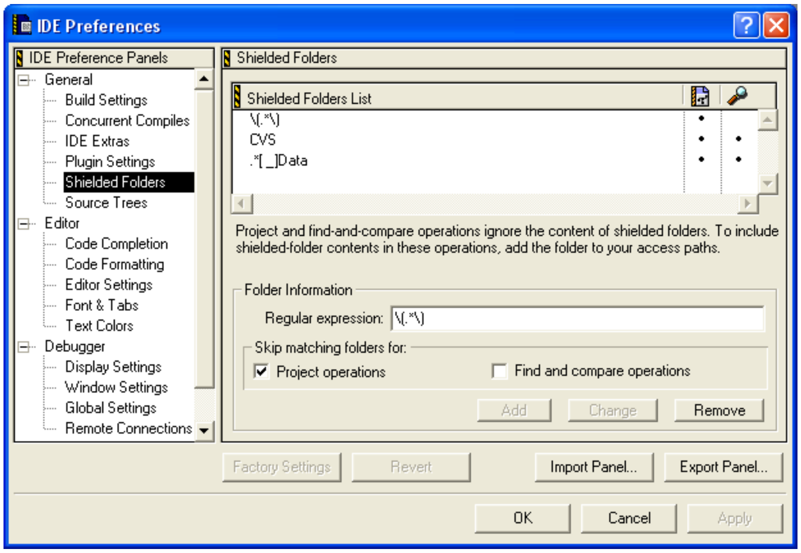
A bullet in the Find-and-compare operations column indicates that the IDE ignores matching folders for find-and-compare operations. These operations include dragging a folder into fields in the Find window, or comparing folder contents.

Note

You can configure a project target to include items within a shielded folder. Use the Access Paths target settings panel to specify the path to the shielded folder that you want to include.

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 settings panel (Figure 9-7).



Shielded Folders List	Project	Find
\(.*\)	•	
CVS	•	•
.*[_]Data	•	•

Project and find-and-compare operations ignore the content of shielded folders. To include shielded-folder contents in these operations, add the folder to your access paths.

Folder Information

Regular expression: \(.*\)

Skip matching folders for:

[✓] Project operations [ ] Find and compare operations

Figure 9-7 Shielded Folders settings panel

2. Enter in the Regular Expression field the expression that specifies the folders you want to ignore. You can enter the name of a specific folder, or you can enter a regular expression that matches several folder names.

3. Choose whether to shield matching folders from project operations, from find-and-compare operations, or from both. Enable the **Project operations** checkbox, the **Find and compare operations** checkbox, or both. During the operations that you specify, the IDE ignores folders with names that match the regular expression.
4. Click **Add**. After you click the Add button, the regular expression appears in the Shielded Folders list. Bullets in the list indicate the operations for which the IDE ignores matching folders.
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 settings panel (Figure 9-6 on page 9-13).
2. Select the regular expression you want to modify.
3. Modify the regular expression using the Regular Expression text box.
4. Specify the operations you want to skip matching folders for, by enabling the **Project operations** checkbox, the **Find and compare operations** checkbox, or both.
5. Click the **Change** button. After you click the Change button, the modified regular expression appears 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 settings panel (Figure 9-6 on page 9-13).
2. Select the regular expression you want to remove.
3. Click the **Remove** button. The IDE deletes the regular expression from the Shielded Folders list.
4. Click **Apply** to apply your changes.

9.3.6 Configuring global source trees

The Source Trees settings panel enables you to define global source trees (root paths) for use in your projects. Source trees are used in dialogs that require you to select a path type, such as the Source Trees panel (shown in Figure 9-8 on page 9-17). 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.

The CodeWarrior IDE predefines four global source tree types:

- absolute path
- project relative
- compiler relative
- system relative.

You can define additional source trees in two panels:

IDE Preferences panel

You can use the source trees you define in the IDE Preferences panel with all projects. This section describes how to configure global source trees.

Target Settings panel

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 10-28 for information on configuring target-specific source trees.

If you define the same source tree in both panels, the target-specific source trees take precedence over the global source trees.

To add, change, or remove 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 panel displays a list of currently-defined source paths.

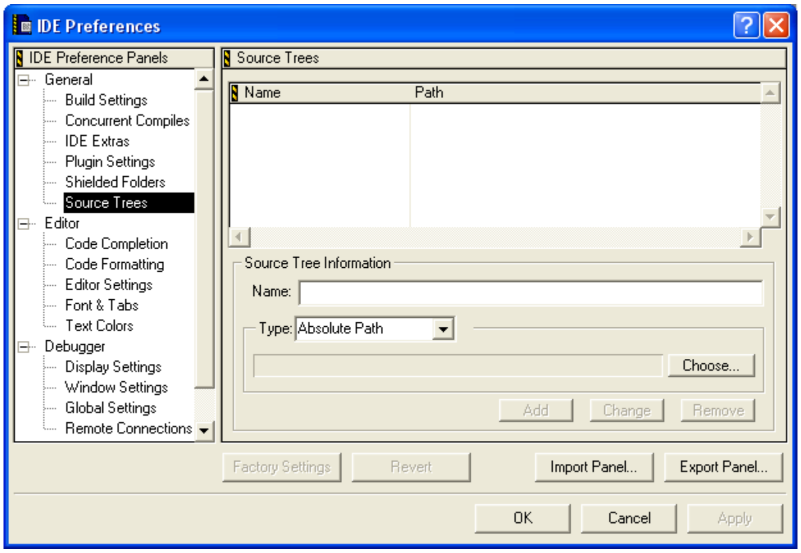


Figure 9-8 Source Trees panel

2. Edit the source tree details:
   - To remove or change an existing source path, double-click the entry in the list of source trees. The source tree details are displayed. Click **Remove** to remove the source tree, or follow the steps below to modify it.
   - To add a new source tree, type a name for the new source path in the Name field.
3. Click the **Type** drop-down list to select the type of source tree. Select one of:

   Absolute Path

   Select this option to choose a specific directory as the root for your source tree.

   Environment Variable

   Select this option to choose a directory defined in an environment variable as the root for your source tree.

   Registry Key

   Select this option to choose a directory defined in a Windows registry key as the root for your source tree.
4. Choose the source tree root:
   - If the source tree is an absolute path, click **Choose…** to select the root directory from the standard file dialog.
   - If the source key is an environment variable, enter the name of the environment variable in the **Name** box and select a type from the **Type** drop-down list. If the environment variable is defined, the source tree window 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 pathname 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 source tree window adds the source tree to the list of defined source trees and displays the value of the registry key.
5. Click one of the following:
   - **Add** if you are adding a new source tree
   - **Change** if you are modifying an existing source tree
   - **Remove** to remove the selected source tree.

6. Click **Apply** to apply your changes. The new source tree name is displayed in dialogs that require you to select a path type, such as the Source Trees panel (shown in Figure 9-8 on page 9-17). See *Configuring access paths* on page 10-12 for more information on adding access paths to CodeWarrior projects.

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 panel. Do this by using the **Export Panel...** button to save a copy of the panel. See *Using the IDE Preferences window* on page 9-3 for information on using the **Export Panel...** button.

9.4 Choosing editor preferences

This section describes the preference panels that control editor features. The editor panels are described in:

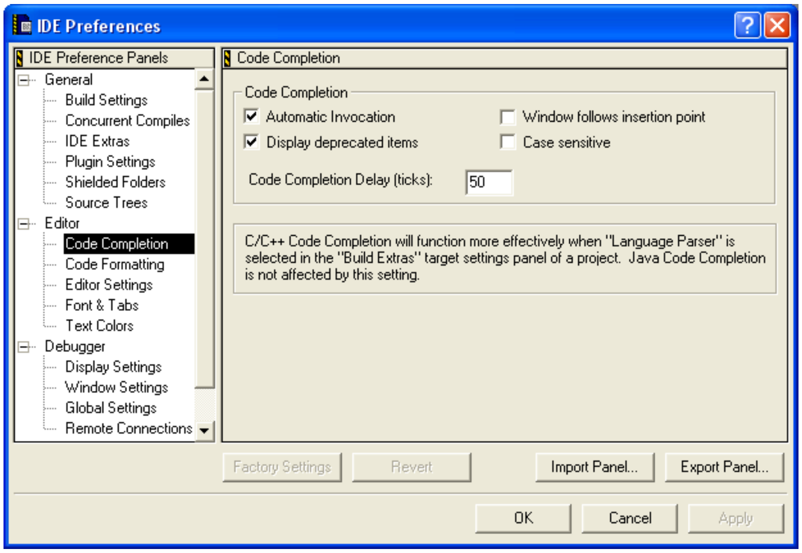
- *Code completion*
- *Code formatting* on page 9-21
- *Editor settings* on page 9-23
- *Font & Tabs* on page 9-28
- *Text Colors* on page 9-31.

9.4.1 Code completion

The Code completion panel enables you to define the behaviour of the CodeWarrior Editor code completion feature which automatically suggests ways to complete source code as it is entered in the Editor window. For more information on Code completion refer to *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).



Code Completion

Code Completion

- [✓] Automatic Invocation
- [✓] Display deprecated items
- [ ] Window follows insertion point
- [ ] Case sensitive

Code Completion Delay (ticks): 50

C/C++ Code Completion will function more effectively when "Language Parser" is selected in the "Build Extras" target settings panel of a project. Java Code Completion is not affected by this setting.

Figure 9-9 Code Completion configuration panel

2. Click the options in the **Code Completion** group box:

   Automatic invocation

   Select this option to automatically open the Code Completion window to complete programming-language symbols. Clear this option to manually open the window.

   Display deprecated items

   Select this option to have the Code Completion window display obsolete items in gray text. Clear this option to have the window hide obsolete 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 IDE consider case when completing code. Clear this option to have the IDE ignore case.

   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. Enter a number in the edit field to specify the delay in ticks, where a tick is 1/60 of a second.

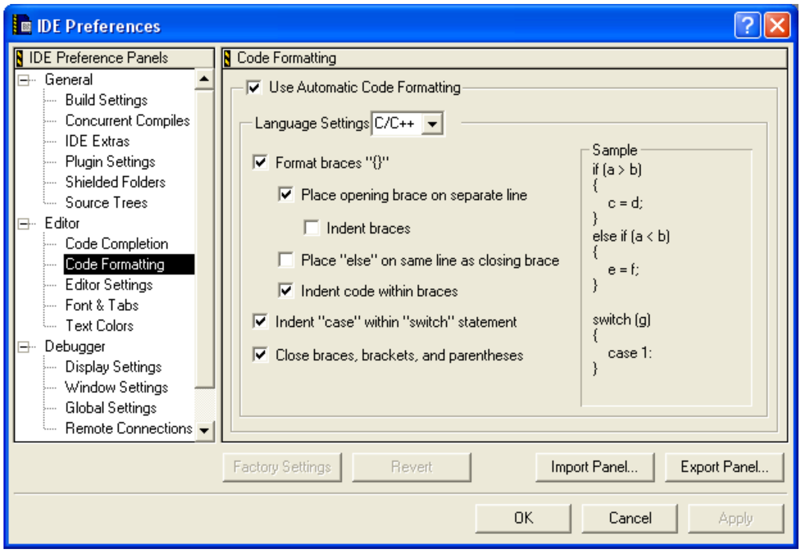
3. Click **Apply** to apply your changes.

9.4.2 Code formatting

The Code formatting panel enables you to configure the automatic formatting style applied to code added in the Editor window. For more information on Code formatting refer to *Formatting code* on page 6-15.

To configure code formatting:

1. Select **Preferences…** from the **Edit** menu and click **Code Formatting** in the IDE Preference Panels list to display the configuration panel (Figure 9-9 on page 9-20).



Code Formatting

☑ Use Automatic Code Formatting

Language Settings C/C++

☑ Format braces "{}"

☑ Place opening brace on separate line

☐ Indent braces

☐ Place "else" on same line as closing brace

☑ Indent code within braces

☑ Indent "case" within "switch" statement

☑ Close braces, brackets, and parentheses

Sample
```
if (a > b)
{
   c = d;
}
else if (a < b)
{
   e = f;
}

switch (g)
{
   case 1:
}
```

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 list box to select the language you wish to configure:
   - C/C++
   - Java

   For each of these languages you can choose various formatting options to apply to code written in this language. As you select the options the code in the **Sample** group box changes to show the formatting effect of the 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 that you typed and the closing brace that it inserts.

   Place brace on separate line

   Select this option to have the editor place an opening brace that you type appear on a separate line.

   Indent braces

   Select this option to have the editor indent braces by one tab stop from the previous line.

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` or `else 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

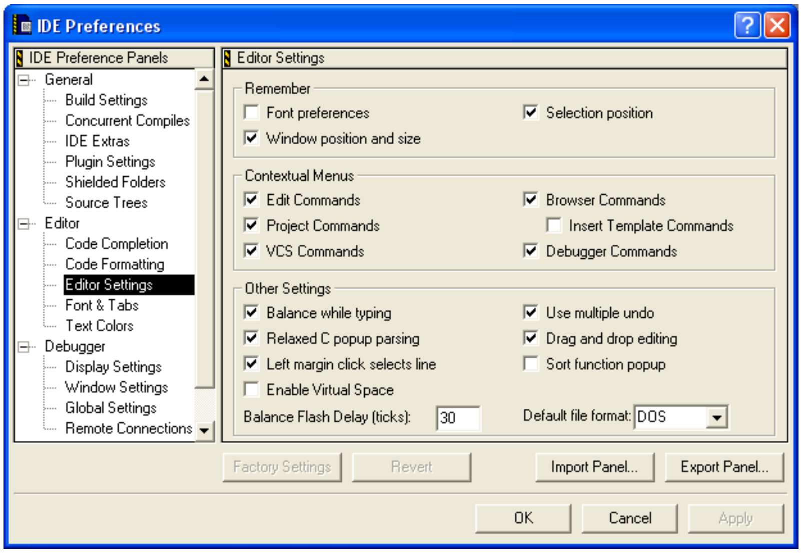
Select this option to have the editor automatically insert the corresponding closing character when you type an opening brace, bracket, or parenthesis. The editor places the cursor between the opening character and the closing character.

4. Click **Apply** to apply your changes.

9.4.3 Editor settings

This section describes how to configure the behavior of the CodeWarrior editor. To open the Editor Settings panel:

1. Select **Preferences…** from the **Edit** menu. The CodeWarrior IDE displays the IDE Preferences dialog.
2. Click **Editor Settings** in the IDE Preference Panels list. The CodeWarrior IDE displays the Editor Settings panel (Figure 9-11 on page 9-24).



Editor Settings

Remember

- Font preferences
- [✓ Selection position]
- [✓ Window position and size]

Contextual Menus

- [✓ Edit Commands]
- [✓ Browser Commands]
- [✓ Project Commands]
- Insert Template Commands
- [✓ VCS Commands]
- [✓ Debugger Commands]

Other Settings

- [✓ Balance while typing]
- [✓ Use multiple undo]
- [✓ Relaxed C popup parsing]
- [✓ Drag and drop editing]
- [✓ Left margin click selects line]
- Sort function popup
- Enable Virtual Space

Balance Flash Delay (ticks): 30

Default file format: DOS

Figure 9-11 Editor Settings preference panel

There are three groups of options. For more information on changing the options see:

- *Setting Remember options*
- *Specifying Contextual Menus* on page 9-25
- *Specifying Other Settings* on page 9-26.

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. Open the Editor Settings panel (see *Editor settings* on page 9-23).
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.

   Window position and size

   Select this option to save the window position and size of editor windows when they are closed.

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.

——— **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 (VCS) and have checked out a read-only copy of a file. (For more information about using the CodeWarrior IDE with a VCS refer to the Metrowerks web site.)

---

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.

Edit Commands

Enable this checkbox to display commonly used menu commands from the Edit menu. See *Edit menu* on page D-5 for more information.

Browser Commands

Enable this checkbox to display commonly used menu commands from the Browser menu. See *Browser menu* on page D-13 for more information. After you enable this checkbox, you can also enable the Insert Template Commands checkbox to display templates from your source code. The contextual menus then display the Insert Template command, as shown in Figure 8-13 on page 8-24. If necessary, this command displays a submenu that lists the templates.

Project Commands

Enable this checkbox to display commonly used menu commands from the Project menu. See *Project menu* on page D-10 for more information.

Insert Template Commands

Enable this checkbox to insert function templates into your source code.

VCS Commands

Enable this checkbox to display commonly used menu commands from the Version Control System (VCS) menu.

Note

The VCS menu is *only* displayed if a version control system plug-in module has been installed. (For more information about using the CodeWarrior IDE with a VCS refer to the Metrowerks web site.)

Debugger Commands

Enable this checkbox to display commonly used menu commands from the **Debug** menu.

Note

The **Debug** menu is not used by the CodeWarrior IDE for RVDS.

Specifying Other Settings

The Other Settings options control how the editor works. To change these settings:

1. Open the **Editor Settings** panel (see *Editor settings* on page 9-23).
2. Select the options you require. The options available are:

   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 Flashing 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 option, **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 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-styled code.

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 & 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. Clicking the window when the mouse pointer faces right selects the line at the mouse pointer. Clicking and dragging the mouse when the mouse pointer faces right selects 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 checkbox 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 editor highlights an opening parentheses, bracket, or brace when balancing punctuation. The Flashing Delay is measured in 60ths 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 text file format** drop-down list to set the end-of-line conventions that the CodeWarrior IDE uses to create new files. You can choose from:

- DOS
- UNIX
- Macintosh.

3. Click **Apply** to apply your changes.

9.4.4 Font & Tabs

The Font & Tabs panel enables you to set the default font and tab information for the CodeWarrior editor. You can change:

- the default settings used by the CodeWarrior IDE for all editor windows
- the settings to be used for an individual file.

——— **Note** ———

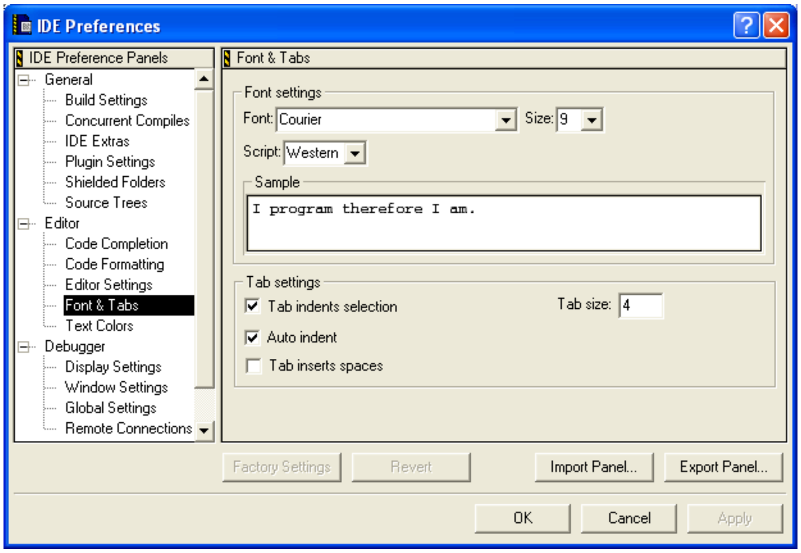
To change settings for individual files, you must ensure that the Remember Font preferences option is selected in the Editor configuration panel. See *Setting Remember options* on page 9-24 for more information. See *Setting the font and tabs for a single file* on page 9-29 for detailed instructions.

———————————

To open the Font & Tabs panel:

1. Select **Preferences…** from the **Edit** menu. The CodeWarrior IDE displays the IDE Preferences dialog.

2. Click **Font & Tabs** in the IDE Preference Panels list. The CodeWarrior IDE displays the Font & Tabs panel (Figure 9-12 on page 9-29).



Font & Tabs

Font settings

Font: Courier Size: 9

Script: Western

Sample

```
I program therefore I am.
```

Tab settings

- [x] Tab indents selection
- [x] Auto indent
- [ ] Tab inserts spaces

Tab size: 4

Figure 9-12 Font & Tabs preference panel

For more information on setting font and tabs options see:

- *Setting the font and tabs for a single file*
- *Setting font and tabs defaults* on page 9-30.

Setting the font and tabs for a single file

To change the font settings for an individual file:

1. Ensure that the **Remember Font** preferences option is selected in the Editor 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. Open the Font & Tabs preference panel (see *Font & Tabs* on page 9-28).

4. Select the display font, font size, and script from the drop-down lists, if required. The font and size you select here are applied to the current editor window.

5. Select tab options if required. The available options are:

   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
- sets the number of characters to be inserted, if the **Tab Inserts Spaces** option is selected.

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.

——— **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.

6. Click **Apply** to apply your changes. The CodeWarrior IDE applies your settings to the current editor file, and uses the 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 (VCS) and have checked out a read-only copy of a file. (For more information about using the CodeWarrior IDE with a VCS refer to the Metrowerks web site.)

Setting font and tabs defaults

To set the default font and tab settings that the CodeWarrior IDE will use for all text files that do not have individual settings specified:

1. Ensure that the **Remember Font** preferences option is not selected in the Editor 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.

9.4.5 Text Colors

The Text Colors preferences panel enables you to change the default text coloring in your source code.

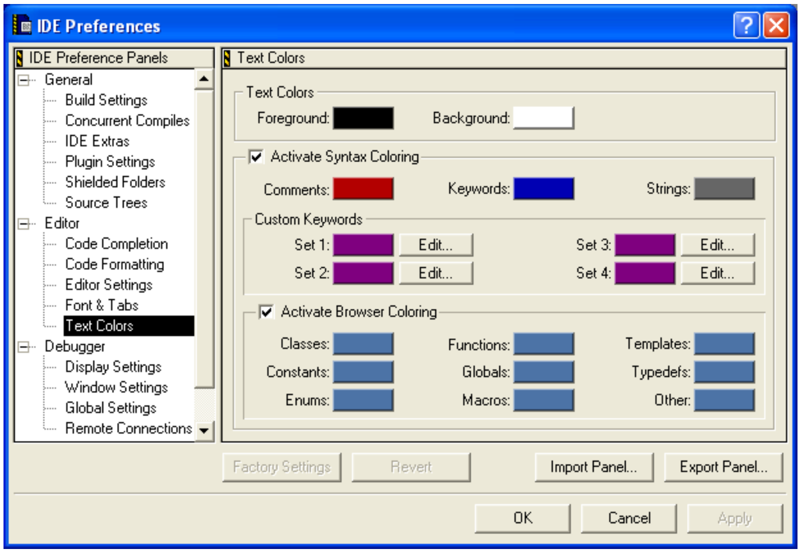
Note

Syntax coloring does not apply to assembly language source. However, you can use browser coloring to highlight assembly language constructs in the browser.

It also provides four Custom Keyword sets that you can use to specify the text color for your own keyword sets. The Custom Keywords list can contain function names, type names, or anything else you want highlighted in your editor windows.

To open the Text Colors panel:

1. Select **Preferences…** from the **Edit** menu. The CodeWarrior IDE displays the IDE Preferences window.
2. Click **Text Colors** in the IDE Preference Panels list. The CodeWarrior IDE displays the Text Colors panel (Figure 9-13).



Text Colors

Text Colors

Foreground: ■ Background: □

[✓] Activate Syntax Coloring

Comments: ■ Keywords: ■ Strings: ■

Custom Keywords

Set 1: ■ Edit... Set 3: ■ Edit...

Set 2: ■ Edit... Set 4: ■ Edit...

[✓] Activate Browser Coloring

Classes: ■ Functions: ■ Templates: ■

Constants: ■ Globals: ■ Typedefs: ■

Enums: ■ Macros: ■ Other: ■

Figure 9-13 Text Coloring preference panel

Changing text colors

In the Text Colors section of the panel, 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 sections. Click the color swatch to change the current color.

Background This option configures the color of the areas on which text appears. Click the color swatch to change the current color.

Changing syntax coloring

The CodeWarrior IDE can use different colors for each type of text. Table 9-1 *Syntax coloring highlights* 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.
Custom keywords	Any keyword listed in the Custom Keyword List. This list is useful for macros, types, and other names that you want to highlight. See *Using color for custom keywords* on page 9-33.

Before you can change the syntax coloring options, you must enable the **Activate Syntax Coloring** checkbox.

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 *Controlling color* on page 6-18 for more information.

Using color for custom keywords

You can use the Custom Keywords section of the Text Colors panel to choose 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.

To add one or more keywords to a Custom Keyword Set:

1. Open the Text Colors panel (see *Text Colors* on page 9-31).
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).

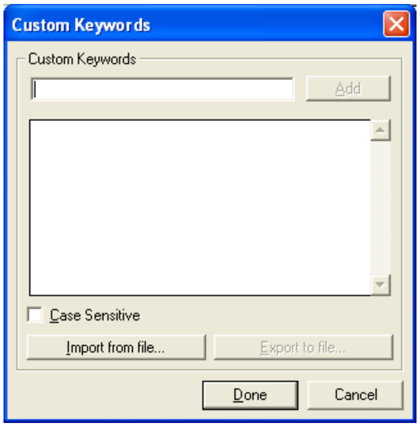


Figure 9-14 Custom Keywords dialog box

3. Type a keyword in the Custom Keywords text field. Check **Case Sensitive** if you want the 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 9-34 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.

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.
   - You can also set target-specific colors for custom keywords. See *Custom Keywords* on page 10-81 for more information.

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 panel (see Figure 9-13 on page 9-31).

2. Click **Edit** to the right of the Custom Keyword Set you want to export or import to (see Figure 9-13 on page 9-31).

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:

   - 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 Syntax Coloring panel to apply the changes to your current editor files.

Changing browser coloring

The Browser Display section of the Text Colors panel enables you to customize the browser. The browser can export its lists of symbols and their types to the CodeWarrior 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** checkbox.

The color choice for each symbol type is displayed in both the editor window and the browser window. Click the color swatch to change the current color.

9.5 Choosing debugger preferences

This panel is not used by the CodeWarrior IDE for RVDS.

9.6 Choosing RAD Tools

This panel is not used by the Code Warrior IDE for RVDS.

9.7 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.

9.7.1 Opening the Customize IDE Commands window

The Customize IDE Commands window contains two tabbed panels that enable you to specify your own commands, assign keybindings to commands, and customize the CodeWarrior IDE toolbar. To open the Customize IDE commands window:

1. Select **Commands & Keybindings…** from the **Edit** menu. The CodeWarrior IDE displays the Customize IDE Commands window (Figure 9-15). The window contains two tabbed panels.

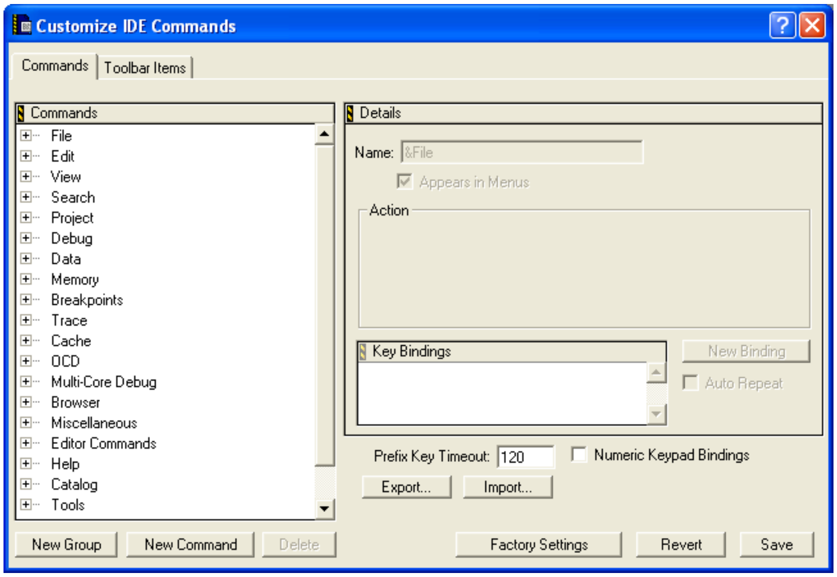


Figure 9-15 Key Bindings panel

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.

- 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.

   ——— **Note** ———

   Some commands are not implemented by the ARM version of the CodeWarrior IDE.

   ————————

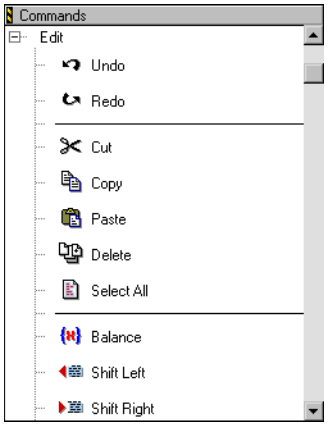


Figure 9-16 List of Edit commands

4. Click on a command to select it. The default key bindings for the command are displayed. See *Customizing keybindings* on page 9-43 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.

9.7.2 Adding your own commands to the CodeWarrior IDE

You can use the **Commands & Keybindings** command in the **Edit** menu to add menu items and keybindings 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 9-38.

2. Create a new group if you want to create a new menu for the command. See *Creating a new command group* on page 9-41 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 Action configuration fields to the Commands window (Figure 9-17).

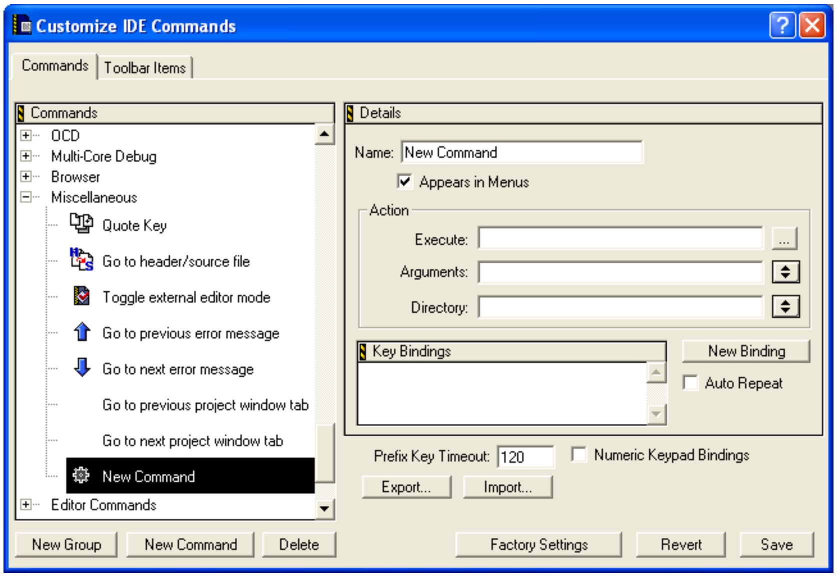
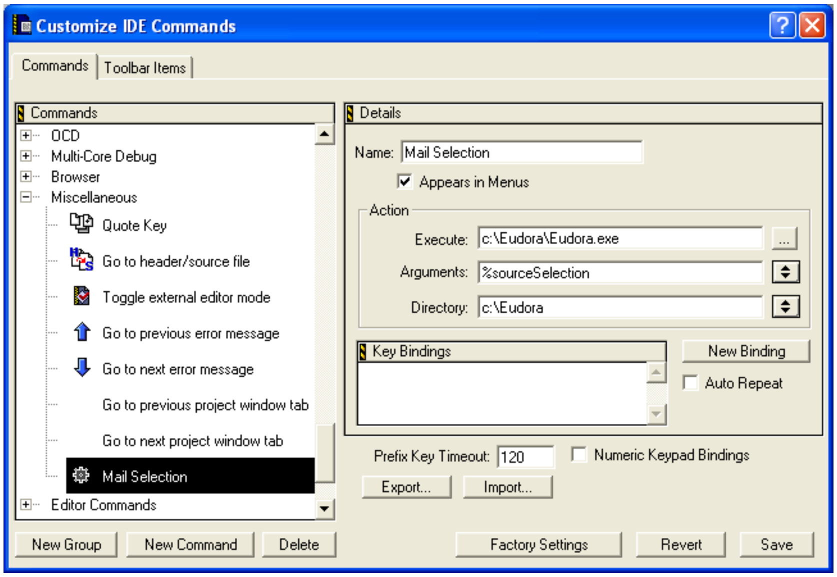


Figure 9-17 Configuring a new command

4. Enter the name of the command to run in the Execute field, or click the **...** button and select the command from the standard file dialog.

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. Figure 9-18 on page 9-41 shows an example.



Name: Mail Selection
[✓] Appears in Menus
Action
Execute: c:\Eudora\Eudora.exe
Arguments: %sourceSelection
Directory: c:\Eudora
[ ] Auto Repeat
Prefix Key Timeout: 120
[ ] Numeric Keypad Bindings

Figure 9-18 Mail selection command

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.

7. Define one or more keybindings for the new command, if required. See *Customizing keybindings* on page 9-43 for more information.

8. Click **Save** to save your settings. The CodeWarrior IDE adds the command to the specified 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 9-38.

2. Click **New Group…**. The CodeWarrior IDE inserts the new group into the commands list (Figure 9-19 on page 9-42).

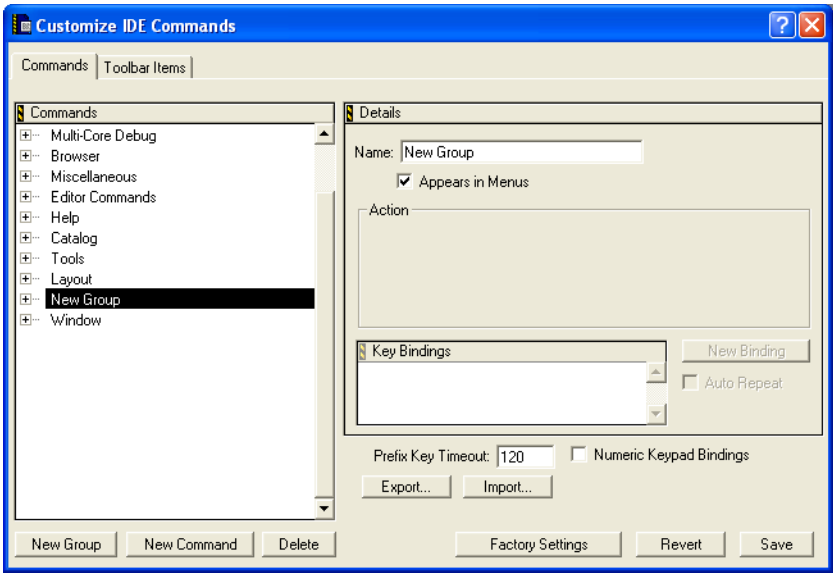


Figure 9-19 Creating a new group

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 9-38.
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.

Note

You only remove commands or groups that you have created, and not those that are pre-defined by the CodeWarrior IDE.

9.7.3 Customizing keybindings

This section describes how to customize the default CodeWarrior IDE keybindings definitions and options. You can customize the keyboard shortcuts used for menu, keyboard, and editor commands in the CodeWarrior IDE. You can attach or *bind* almost any key to any command, and you can define multiple keybindings for the same command. You can set the key bindings for menu commands, source code editor actions, and other miscellaneous actions. You can also create multiple-keystroke command bindings.

This section describes:

- *Restrictions on choosing key bindings*
- *Using multiple-keystroke bindings*
- *Setting the Prefix Key Timeout* on page 9-44
- *Using a Quote Key prefix to create single-key keybindings* on page 9-44
- *Setting Auto Repeat for keybindings* on page 9-46
- *Adding a new keybinding* on page 9-46
- *Deleting a keybinding* on page 9-47
- *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:

- The Escape and Space keys are always invalid for key bindings.
- Function keys and the Clear key are valid for creating key bindings.
- The Return and Tab keys require at least the Control or Shift key. This restriction does not apply for the second key of a two-key sequence.

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 Control-X followed by Control-S.

To emulate the Emacs key binding to save a file:

1. Delete the Ctrl-X keybinding for the **Cut** command. You must delete the current keybinding because you cannot assign the same keybinding to more than one command.
2. Delete the Ctrl-S keybinding for the **Save** command.
3. Set the command key for the **Save** command to Control-X Control-S.

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 multi-keystroke binding is pressed. Larger values indicate that the CodeWarrior IDE will wait longer for the second key to be pressed.

To set the Prefix Key timeout:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).
2. Enter the value for the timeout key in the **Prefix Key Timeout** text field.

   The timeout value is in *ticks* (1/60th of a second). Valid values are in the range of 1 to 999. The default value is 120.
3. Click **Apply** to save your settings.

Using a Quote Key prefix to create single-key keybindings

In typical use, a key binding requires you to use two keys in combination:

- a modifier key, such as the Control 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 key for the number 1, with no modifier, to a command.

If you assign a keybinding to a single printing key, you must type a Quote Key prefix in order to ignore the keybinding 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 on-screen.

By default, the CodeWarrior IDE does not define a Quote Key. To assign a Quote key:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).

2. Click the hierarchical control next to the Miscellaneous group and select the Quote Key entry (Figure 9-20).

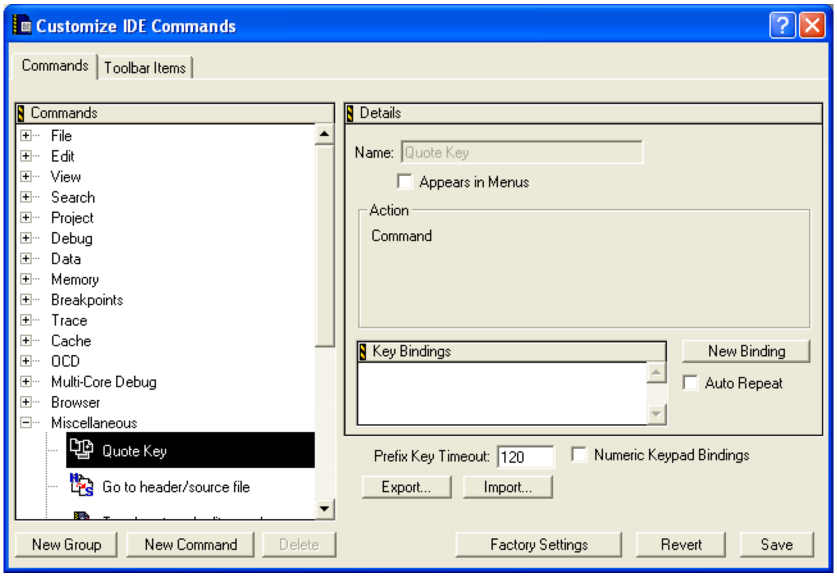


Figure 9-20 Selecting the quote key

3. Click New Binding to display the Edit Key Binding dialog (Figure 9-21).

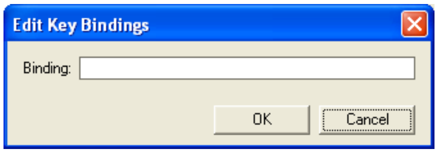


Figure 9-21 New Key binding for the Quote key

4. Type the key you want to use as the quote key prefix and click **OK** to set the key binding.

5. Click **Apply** to apply your settings.

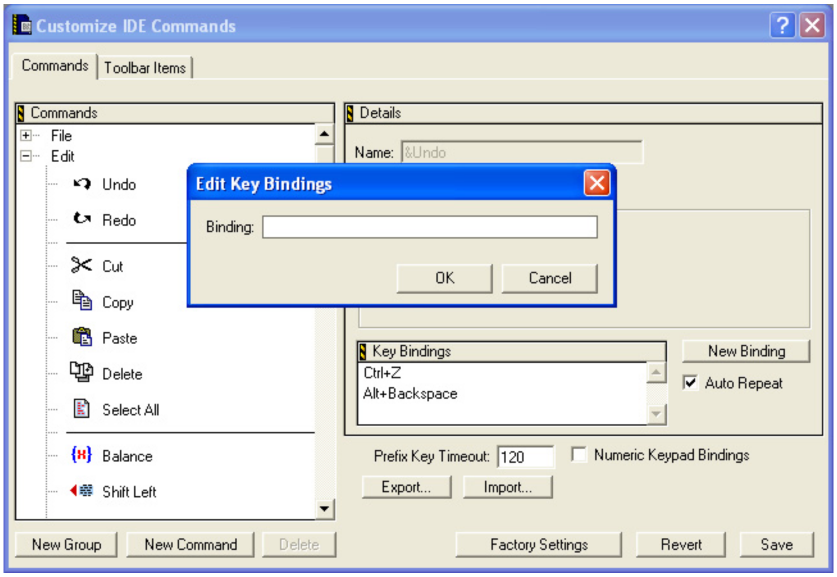
Setting Auto Repeat for keybindings

You can use Auto Repeat to specify that a keybinding 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. You can configure Auto Repeat separately for each key binding.

Adding a new keybinding

To add a new keybinding for a command:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).
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 Keybinding dialog (Figure 9-22).



Name: &Undo

Binding:

Key Bindings: Ctrl+Z, Alt+Backspace

[✓] Auto Repeat

Prefix Key Timeout: 120

[ ] Numeric Keypad Bindings

Figure 9-22 Edit Keybinding dialog

4. Type the key sequence you want to use for the command, and click either:
   - **OK** to confirm your setting
   - **Cancel** if you make a mistake.
5. Click **Apply** to apply your settings.

Deleting a keybinding

To delete a keybinding:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).
2. Select the keybinding you want to delete.
3. Press the Delete key. The keybinding is deleted from the list of keybindings.
4. Click **Apply** to apply your settings.

Exporting key bindings

You can save your key bindings in a file so that you can later import them into the CodeWarrior IDE at another time. To export your current key bindings:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).
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 keybindings are saved to the keybindings file.

Importing key bindings

To import a key bindings set that you have previously saved:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 9-38).
2. Click **Import** and select the keybindings file from the standard file dialog.
3. Click **Open** to read the keybindings from the file.

9.8 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 9-49.

9.8.1 Toolbar overview

A toolbar contains elements, represented by icons, that 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**, **Function**, **Header**, **Marker**, **Version Control**, and **Current Target** drop-down lists.

Figure 9-23 shows the default toolbar from the CodeWarrior IDE project window.

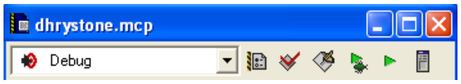


Figure 9-23 The Project window toolbar

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 9-49 for more information on adding and removing elements to toolbars.

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.

9.8.2 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:

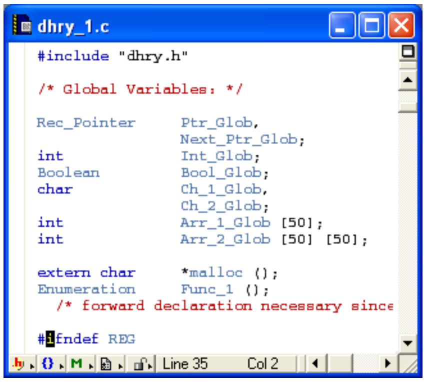
1. Click on the window which you want to configure to make it the active window.
2. Select either:
   - **View** → **Toolbars** → **Hide Window Toolbar** to hide the window toolbar
   - **View** → **Toolbars** → **Show Window Toolbar** to show the window toolbar.

   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 elements are displayed at the bottom of the editor window. Figure 9-24 shows an example.

   ————————————



```
#include "dhry.h"

/* Global Variables: */

Rec_Pointer       Ptr_Glob,
                  Next_Ptr_Glob;
int               Int_Glob;
Boolean           Bool_Glob;
char              Ch_1_Glob,
                  Ch_2_Glob;
int               Arr_1_Glob [50];
int               Arr_2_Glob [50] [50];

extern char       *malloc ();
Enumeration       Func_1 ();
  /* forward declaration necessary since

#ifndef REG
```

Figure 9-24 The editor window with hidden toolbar

9.8.3 Modifying a toolbar

You can modify a toolbar by:

- *Adding a toolbar element* on page 9-50
- *Removing a toolbar element* on page 9-51
- *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 9-51.

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 & Keybindings** from the **Edit** menu. The Customize IDE Commands window is displayed. The window contains two tabbed panels:

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 files 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 (Figure 9-25 on page 9-51).

   ——— Note ———

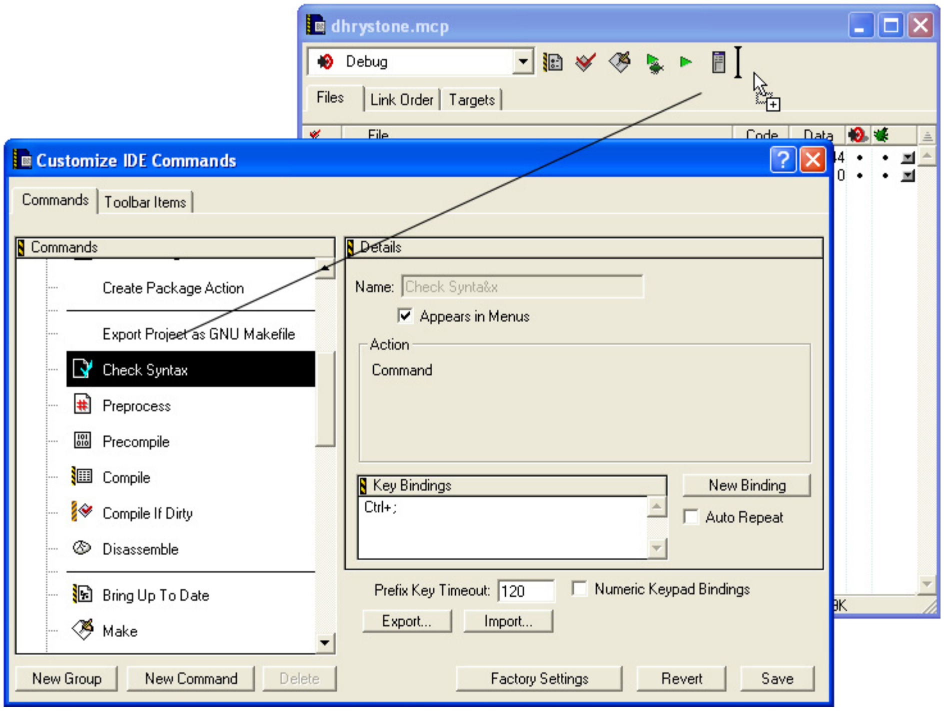
   You must click on the icon. You cannot drag the element if you click on its name.

   ———————

   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 Dirty and File path indicator in the Toolbar Controls tab can only be added to the editor window toolbar

- the **Target** pop-up element in the Toolbar Controls tab can only be added to the project window toolbar.



Name: Check Syntax
[✓] Appears in Menus
Key Bindings: Ctrl+;
Prefix Key Timeout: 120

Figure 9-25 Dragging a toolbar element

5. Release the mouse button to add the element to the toolbar.

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. Select either:
   - **Window → Toolbar → Clear Window Toolbar** to clear all toolbar items from they type of toolbar in the currently selected window
   - **Window → Toolbar → Clear Main Toolbar** to clear all toolbar items from the main window toolbar.

   Note

   Some default toolbar items cannot be removed.

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. Select either:
   - **Window → Toolbar → Reset Window Toolbar** to reset the type of toolbar in the currently active window
   - **Window → Toolbar → Reset Main Toolbar** to reset the main window toolbar.

   The toolbar is reset to contain its default elements.

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:

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 set the Target options 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 the:

- Debug build target
- Release build target.

See Chapter 4 *Working with the ARM Project Stationery* for more information on default ARM stationery. See also *Setting the current build target* on page 3-46 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. For example, one panel contains settings that specify the folders in which the CodeWarrior IDE searches for the source files and libraries. See Figure 10-2 on page 10-6 for an example of the Target Settings window. 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 tool chain. 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 linker 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 editor.

Debugger The panels in this group enable you to select the ARM 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.

Miscellaneous

There is only one panel in this group which 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
- the RealView linker, 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 panels for the RealView Compiler and RealView Assembler, to ensure that the settings are appropriate to your target hardware and procedure call standard preferences
- the RealView Linker panels, to ensure that the appropriate output is produced
- the RealView `fromelf` panels, if you are using `fromelf` to produce a ROMable image 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 4-32 for more information.

10.2 Overview of the Target Settings window

This section gives an overview of how to use the Target Settings window to set 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 10-35
- *Configuring linker settings* on page 10-64
- *Configuring editor settings* on page 10-81
- *Configuring the debugger* on page 10-83
- *Configuring Miscellaneous features* on page 10-91.

——— **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. See the Via File Syntax Chapter of the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for information on using via files.

————————

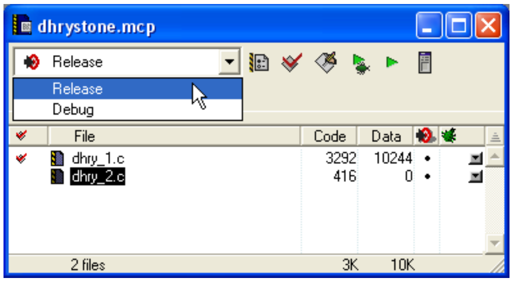
10.2.1 Using the Target Settings window

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 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 3-15 for more information.
2. Use the **build target** drop-down list to select the build target you want to configure (Figure 10-1 on page 10-5).



File	Code	Data
dhry_1.c	3292	10244
dhry_2.c	416	0
2 files	3K	10K

Figure 10-1 Select build target

3. Either:
   - click the **Target Settings** button in the Project window.
   - select the **Settings** menu item for your build target from the **Edit** menu.

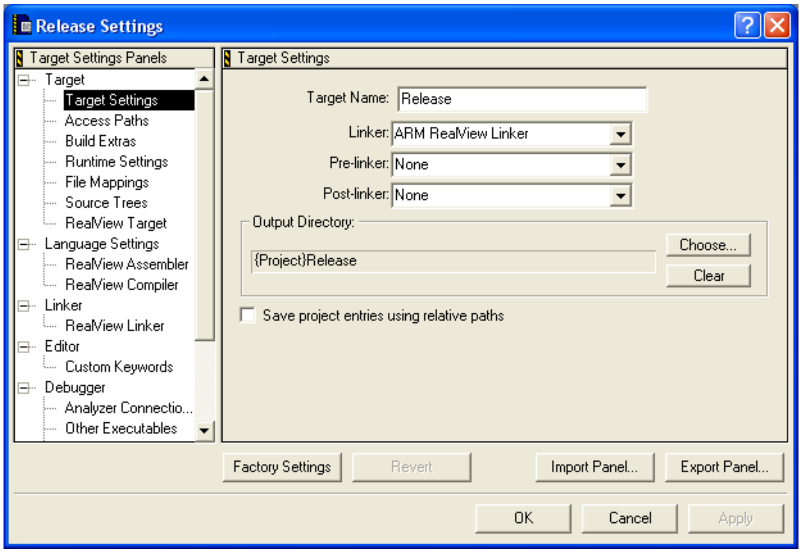
     The name of the **Settings** menu item matches the build target for your currently selected project. For example, if your current build target is Debug, the **Settings** menu item is named **Debug Settings…**.

   The CodeWarrior IDE displays a Target Settings window with a list of available 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.

   ————————



Target Settings

Target Name: Release

Linker: ARM RealView Linker

Pre-linker: None

Post-linker: None

Output Directory:

{Project}Release

Choose...

Clear

Save project entries using relative paths

Figure 10-2 Selecting a 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.

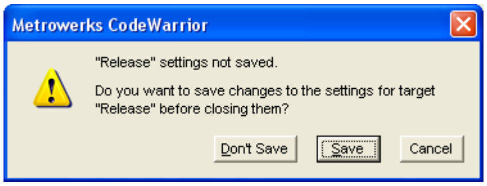
   Each panel contains related options that you can set. The options you select apply to the currently selected build target in the active project.

5. Select the options you require. See the following sections in this chapter for detailed descriptions of the options in each configuration panel.

6. Save or discard your changes, as required. See *Applying or discarding changes* for more information on applying the changes you have made.

Applying or discarding changes

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).


Metrowerks CodeWarrior

"Release" settings not saved.

Do you want to save changes to the settings for target "Release" before closing them?

Figure 10-3 Settings Confirmation dialog box

Click one of:

- **Don't Save** to discard your changes and close the dialog box
- **Save** to save your changes and close the dialog box
- **Cancel** to continue using the Target Settings window without saving changes.

In addition, you can use the dialog buttons in the Target Settings window to apply or discard your changes. The dialog buttons are:

OK Click this button to save any changes that you have made and close the panel.

Cancel Click this button to close the panel 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.

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.

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 to use. It gives information on:

- *Configuring target settings*
- *Configuring access paths* on page 10-12
- *Configuring build extras* on page 10-20
- *Configuring runtime settings* on page 10-23
- *Configuring file mappings* on page 10-24
- *Configuring source trees* on page 10-28
- *Configuring the RealView target* on page 10-30.

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 panel enables you to specify basic settings for your project such as:

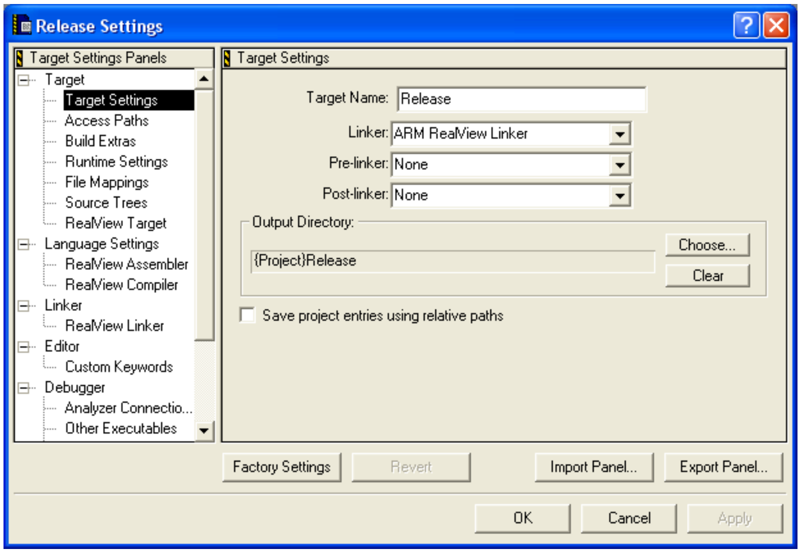
- the name of the target
- the linker and post-linker to use
- the output directory, and whether to use relative or absolute paths for files in the project.

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. As you select certain options from the Target Panels list you will notice that other panels will be added and removed from the panels list as appropriate.

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 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).


Release Settings

Target Settings Panels

- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - Runtime Settings
  - File Mappings
  - Source Trees
  - RealView Target
- Language Settings
  - RealView Assembler
  - RealView Compiler
- Linker
  - RealView Linker
- Editor
  - Custom Keywords
- Debugger
  - Analyzer Connectio...
  - Other Executables

Target Settings

Target Name: Release

Linker: ARM RealView Linker

Pre-linker: None

Post-linker: None

Output Directory: {Project}Release

Save project entries using relative paths

Figure 10-4 Target Settings 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 4-39 for more information on how the CodeWarrior IDE names output files.

Linker Specify the linker to use.

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**.

Select a linker option from the drop-down list. You can choose:

None No link stage is required for the target. This means that source files will not be compiled or assembled, because your choice of linker determines the compiler and assembler that the CodeWarrior IDE calls. You can use this option if you want to define a pre-link or post-link step only, or 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 10-65 for more information on linker options.

ARM RealView Librarian

Use the ARM RealView Librarian to create a library from the compiler and linker output.

Note

See *Configuring file mappings* on page 10-24 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. Choose one of:

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 10-75 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 panel.
- the output type is set to **Linker Output** in the RealView Target panel.

When you select this option, a **RealView FromELF** configuration panel is added to the **Linker** panels list.

Batch File Runner

Use the Batch File Runner 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 runner* on page 4-51 for detailed information on using the Batch File Runner.

Output Directory

This field displays the name of the directory where the data directory containing your build output is placed. The default location is the directory that contains your project file. 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 directory specified does not exist when you open the project, then 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 if it needs to re-search for files in the access paths. If you use relative paths, then 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 allow you to add multiple copies of source files that produce output objects. See *Filename requirements* on page 3-29 for more information.

————————

4. Click **Apply** to apply your changes.

10.3.2 Configuring access paths

You can use the **Access Paths** panel to define directories that the CodeWarrior IDE searches for libraries, header files, and source files. The CodeWarrior IDE defines two types of access path:

User Paths

By default, the User path is set to the main location of your project. The RealView tools search these paths for:

- User header files. These are header files that you include with a `#include"…"` header files statement.
- User libraries. These are libraries that correspond to your `#include"…"` library statements.
- Your source files. When you add a source file to your project from any directory, the CodeWarrior IDE adds the path for the source file to the User Paths list automatically.

By default, the User Paths setting contains `{Project}`. This is the folder that contains the open project.

System Paths

The RealView tools search these paths for standard header files and libraries. When you create a project using ARM stationery, the system path list contains:

- `{RVCT22INC}`. The default directory for the ARM standard C++ library header files. (This system path refers to the same directory as the `$RVCT22INC` environment variable which is created when RVDS is installed.)

Unless you are working with you own C libraries you should not need to change these paths. For more information about these variables and how they are used by the RVCT, refer to the *RealView Compilation Tools v2.2 Compiler and Libraries Guide*.

Default header file search paths

——— **Note** ———

- 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 compiler when they are invoked from the command line. In particular:
  - — the `:mem` directory is not searched
  - — there is no *current place* (Berkeley 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 *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for details of how the compiler searches for header files when invoked from the command line.

- 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* for more information.

You can change the default search paths in the following ways:

- 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` of a panel. If you do, CodeWarrior for RVDS has no knowledge of the included files and you lose the Browse information and Error processing associated with the included files. Instead, add the include file paths using the Access Paths panel for the build target as described in *Adding an access path* on page 10-16.

If your standard header files or libraries 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.

Setting access path options

There are a number of options you can set to modify the way in which 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 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 on page 10-14).

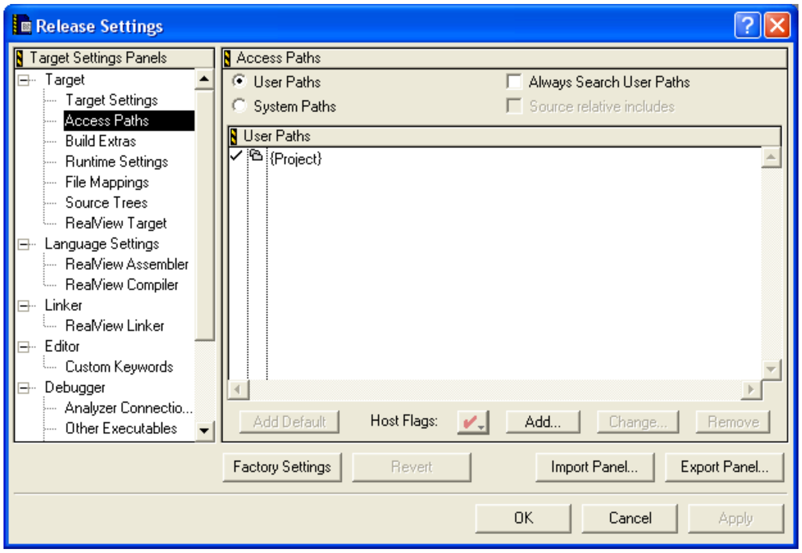


Figure 10-5 Access Paths settings panel

3. Select one of:
   - **User Paths**
   - **System Paths**

   depending on which path type you want to modify.

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 3-42 for more information.

   Click in the Recursive Search column next to a folder name to toggle searching recursively through the folder (see Figure 10-6 on page 10-15):

   - If a folder icon is displayed next to the name of the folder, the CodeWarrior IDE performs a recursive search on the path. That is, the CodeWarrior IDE searches that folder and all the folders within it.
   - If a folder icon is not displayed, the CodeWarrior IDE searches the named folder only.

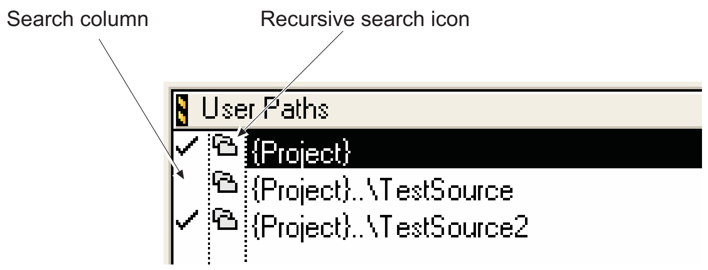


Figure 10-6 Access paths detail

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 System path list or the User path.

Disabling search for a directory

Click in the Search column next to a folder name to toggle searching that directory (see Figure 10-6):

- if a check mark is displayed next to the name of a folder, the folder is searched.
- if a check mark is not displayed, the folder is not searched.

Note

You can prevent any folder and all its subfolders in an access path from being searched by renaming the Windows directory with enclosing parentheses. For example, changing `GameImages` to `(GameImages)` excludes the folder 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 panel. 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 10-12 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 3-42 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.

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.

To add a new access path to a project:

1. Display the Target Settings 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:
   - **User Paths**
   - **System Paths**

   depending on which path type you want to add.
4. Click **Add**. The CodeWarrior IDE displays the Select Access Path dialog (Figure 10-7 on page 10-17).

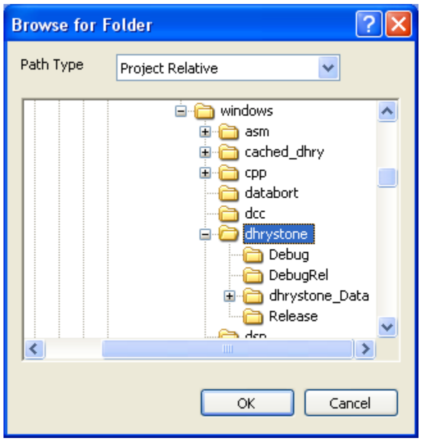


Figure 10-7 Select an Access Path dialog box

5. Select the path type you require from the **Path Type** drop-down list.

——— **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 will slow performance.

———————

You can select from the following path types:

Absolute Path

The CodeWarrior IDE defines the access path of the added folder relative to the root level of the startup volume, including all folders 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 folders along the access path.

Project Relative

The CodeWarrior IDE defines the access path of the added folder relative to the folder that contains the project. You do not need 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 folder 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 folder relative to the folder that contains the CodeWarrior IDE executable. You do not need 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 folder 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 folder relative to the base folder containing your operating system. You do not need 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 folder on a volume other than the one on which your active operating system base folder resides.

Source Tree Relative

The **Path Type** drop-down list 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 folder relative to a folder defined in either a build target source tree, or a global source tree. See *Configuring source trees* on page 10-28 for more information on defining source trees for a specific build target. See *Configuring global source trees* on page 9-16 for more information on defining source trees for all projects.

6. Select the folder 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 also 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.

Changing or removing an access path

To change an access path for a build target:

1. Display the Target Settings 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:
   - **User Paths**
   - **System Paths**

   depending on which path type you want to change or remove.
4. Select the path you want to change or remove and click either:
   - **Change**, if you want to change the access path. The CodeWarrior IDE displays the Select Access Path dialog (see Figure 10-7 on page 10-17). Use the dialog box to navigate to the new folder location. See *Adding an access path* on page 10-16 for a description of the Path Type options.
   - **Remove**, if you want to remove the access path. 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 Access paths lists.

   ———————————
5. Click **Apply** to apply your changes.

Adding the default access path

You can click the **Add Default** button to add the default CodeWarrior IDE paths to the User and System paths lists if you delete them by accident.

——— **Note** ———

If you are working with a project based on ARM stationery, the default System path for the project is:

- `{RVCT22INC}`

The **Add Default** button does not add this path to your project. It adds the CodeWarrior IDE default path of recursive `{Compiler}`. You must add the ARM-defined default path yourself if you delete it. See *Adding an access path* on page 10-16 for information.

Host Flags

The **Host Flags** drop-down list specifies the host platform that can use an access path. This menu does not apply to the ARM version of the CodeWarrior IDE. By default, all host platforms are selected. 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.

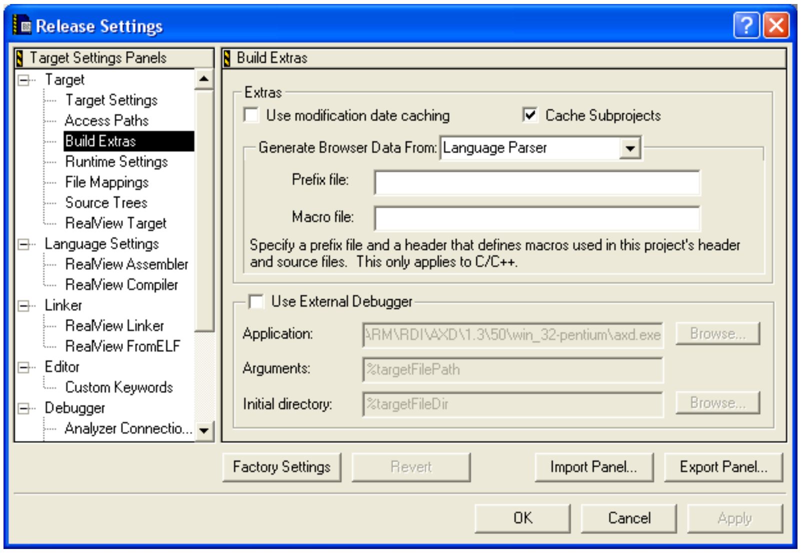
10.3.3 Configuring build extras

The **Build Extras** panel contains a number 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 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 (Figure 10-8 on page 10-21).



Build Extras

Extras

☐ Use modification date caching ☑ Cache Subprojects

Generate Browser Data From: Language Parser

Prefix file:

Macro file:

Specify a prefix file and a header that defines macros used in this project's header and source files. This only applies to C/C++.

☐ Use External Debugger

Application: ARM\RDI\AXD\1.3\50\win_32-pentium\axd.exe

Arguments: %targetFilePath

Initial directory: %targetFileDir

Figure 10-8 Build Extras settings 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 of CodeWarrior, then CodeWarrior is unable to detect any changes to those sources.

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 3-42 for more information on using the CodeWarrior IDE in a complex build environment.

Cache Subprojects

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 subprojects. 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

Select this option if you want the CodeWarrior IDE to generate symbolics 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 IDE editor.

- 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 editor, in addition to all browser windows. You should consider generating 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 checkbox if you are using a third-party debugger instead of the ARM debuggers. You must specify the following options:

Application

Click **Browse...** to search for the non-ARM 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 is run from within the CodeWarrior IDE, enter them in this field. See the documentation accompanying the non-ARM 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.

10.3.4 Configuring runtime settings

Not supported in CodeWarrior for RVDS.

10.3.5 Configuring file mappings

Use the File Mappings settings 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
- the currently selected build target.

The default file mappings for ARM stationery projects are shown in Table 10-1:

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
`.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

Table 10-1 File mappings (continued)

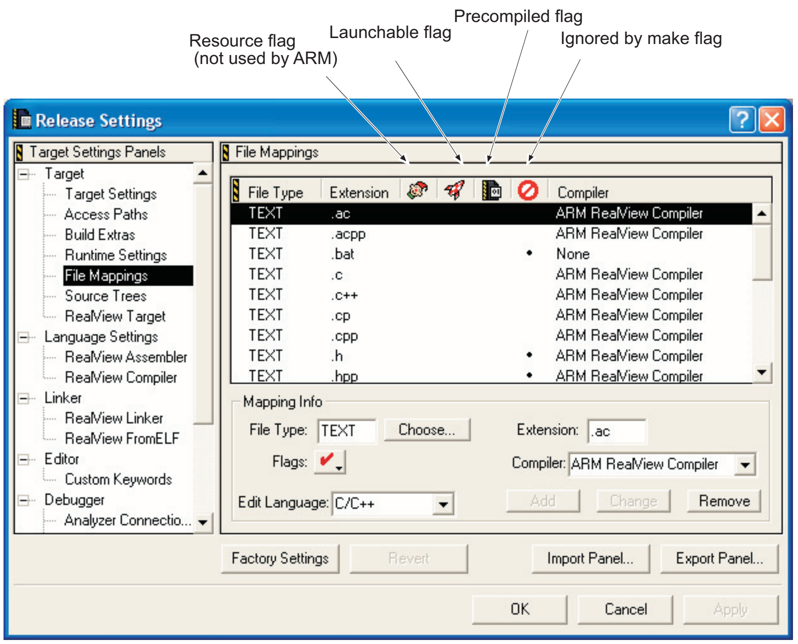
Extension	File Type	Compiler	Description
`.a` or `.lib`		ARM ELF Importer	ARM library file
`.axf`		ARM ELF Importer	ARM image
`.sym`		ARM ELF Importer	A file containing symbol definitions
`.tdv`		ARM ELF Importer	Intermediate build file for a subproject
`.txt`	Text	None	Text file
`.bat`	Text	None	DOS batch command file for post linker
`.scat`	Text	None	Scatter-loading file

Note

- File mappings determine whether the CodeWarrior IDE will recognize files in the project. If you have trouble adding files to your project, or if the CodeWarrior IDE refuses a folder or file that is dragged and dropped on the Project window, check the File Mappings settings 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.

To view and modify the file mappings for a build target:

1. Display the Target Settings 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 10-26).



Target Settings Panels
- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - Runtime Settings
  - File Mappings
  - Source Trees
  - RealView Target
- Language Settings
  - RealView Assembler
  - RealView Compiler
- Linker
  - RealView Linker
  - RealView FromELF
- Editor
  - Custom Keywords
- Debugger
  - Analyzer Connectio...

File Mappings

File Type	Extension	Resource	Launchable	Precompiled	Ignored by make	Compiler
TEXT	.ac					ARM RealView Compiler
TEXT	.acpp					ARM RealView Compiler
TEXT	.bat				•	None
TEXT	.c					ARM RealView Compiler
TEXT	.c++					ARM RealView Compiler
TEXT	.cp					ARM RealView Compiler
TEXT	.cpp					ARM RealView Compiler
TEXT	.h				•	ARM RealView Compiler
TEXT	.hpp				•	ARM RealView Compiler

Mapping Info

File Type: TEXT   Choose...   Extension: .ac
Flags: ✓   Compiler: ARM RealView Compiler
Edit Language: C/C++   Add   Change   Remove

Factory Settings   Revert   Import Panel...   Export Panel...
OK   Cancel   Apply

Figure 10-9 File Mappings 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 the **Choose…** button and select a file of the same type.

4. Enter the filename extension, such as `.c` or `.h`, for the file type.

5. Click the **Flags** drop-down list to set flags that determine how the CodeWarrior IDE treats files of the current type. If a flag is set, the File Mappings panel displays a dot in the appropriate flag column (see Figure 10-9). The flags are:

   Resource File

   This flag does not apply to the RVDS tool chain.

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.

Precompiled

Select this flag to instruct the CodeWarrior IDE to compile this type of file before other files. This option is useful for file types 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 list box and select a compiler for the File Type 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 list to select the language you want to edit in. Selecting the correct language ensures that the correct syntax coloring for your chosen language is used.
8. Click either:
   - the **Add** button, to add a new file mapping
   - the **Change** button, to change the configuration for a selected file mapping.
9. Click **Apply** to apply your changes.

To remove a file mapping:

1. Display the Target Settings 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 10-26).
3. Click the entry you want to modify in the **File Mappings** list.
4. Click the **Remove** button to remove the selected file mapping.

10.3.6 Configuring source trees

The Source Trees settings 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 define your access paths using source trees it makes your projects far easier to move to another computer and also facilitates sharing of sources between developers.

You can define source trees in two panels:

Target Settings Panel

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 Preferences Panel

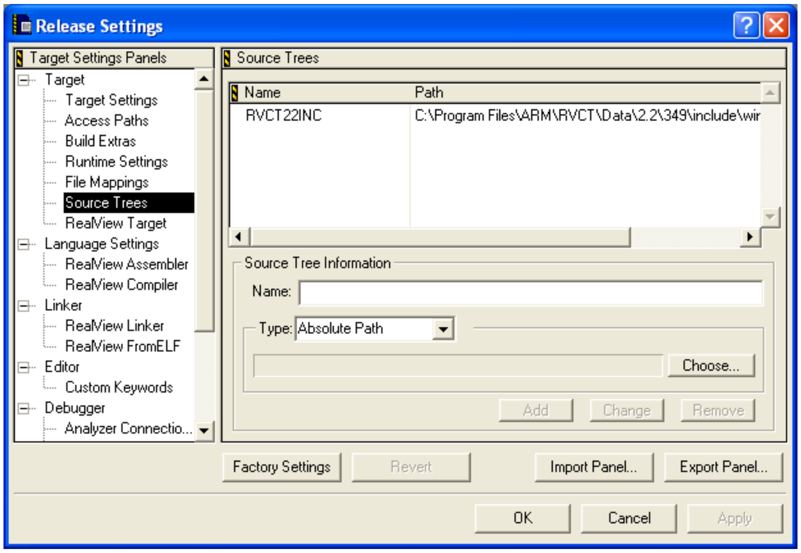
You can use the source trees you define in the IDE Preferences panel with all projects and build targets created using CodeWarrior. See *Configuring global source trees* on page 9-16 for information on configuring global source trees.

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, the environment variable `RVCT22INC` 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 `RVCT22LIB` environment variable.

To add, change, or remove a source tree for the current build target:

1. Display the Target Settings 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 (Figure 10-10 on page 10-29). The source trees panel displays a list of currently defined source paths.



RVCT22INC C:\Program Files\ARM\RVCT\Data\2.2\349\include\wir

Type: Absolute Path

Figure 10-10 Source Trees panel

3. Edit the source tree details:
   - To remove or change an existing source path, click on the entry in the list of source trees. The source tree details are displayed in the Source Tree information group box. Click **Remove** to remove the source tree, or follow the steps below to modify it.
   - To add a new source tree, type a name for the new source path in the Name field and follow the steps below.

4. Click the **Type** drop-down list to select the type of source tree. Select one of:

   Absolute Path

   Select this option to choose a specific directory as the root for your source tree.

   Environment Variable

   Select this option to choose a directory defined in an environment variable as the root for your source tree.

   Registry Key

   Select this option to choose a directory defined in a Windows registry key as the root for your source tree.

5. Choose the source tree root:
   - If the source tree is an absolute path, click **Choose…** to select the root directory from the standard file dialog.

- If the source key is an environment variable enter the name of the environment variable. If the environment variable is defined, the source tree window 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 pathname 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 source tree window 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
   - **Remove** to remove the selected source tree.

7. Click **Apply** to apply your settings. The new source path is displayed in dialogs that require you to select a path type, such as the Select an Access Path dialog. For example, Figure 10-11 shows an entry for RVCT22BIN. (See *Configuring access paths* on page 10-12 for more information on adding access paths to CodeWarrior projects.)

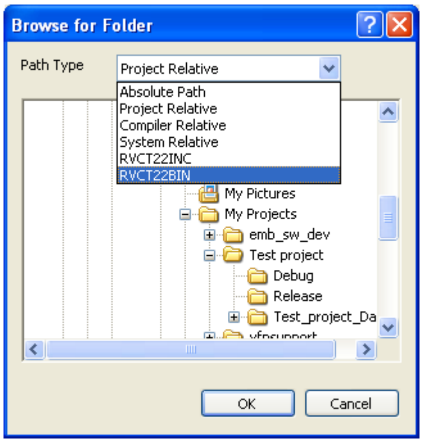


Figure 10-11 Example Select Access path dialog box

10.3.7 Configuring the RealView target

The RealView Target panel enables you to change the output name and output type for a project.

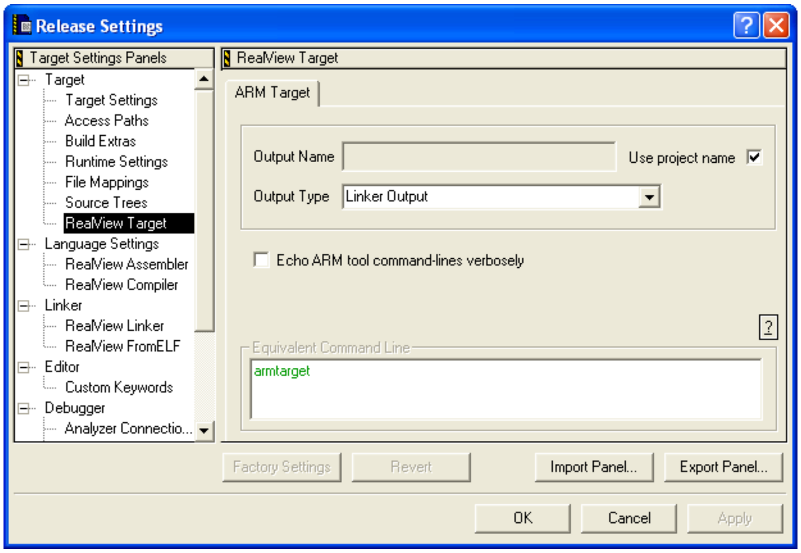
The default output filename is the same as the project name. For example, a project called `test` might produce the image file `test.axf`. You can use this panel to produce a different image file, for example, `new.axf`, without changing the name of the project. You might want to change the name of a project if the build target is to be a subtarget 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 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 (Figure 10-12). The RealView target panel displays the current settings for the output name and output type.



Output Name
Use project name ✓
Output Type: Linker Output
Echo ARM tool command-lines verbosely
Equivalent Command Line
armtarget

Figure 10-12 RealView Target 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 list to select the output type. Select one of:

   Linker Output

   Select this option if you are using the RealView linker or librarian. This option is the default.

   The output produced will depend on the Linker setting in the Target Settings panel.

   - if the Linker is set to **ARM RealView Linker**, then 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**, then 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.

   When this option is set, only object files are produced. This is the usual setting if the build target is to be a subtarget of another build target. The subtarget generates the object files, and the parent build target links the object files to create an executable image. For example, the ARM subtarget of the Thumb ARM Interworking Image stationery (see *Using the Thumb ARM interworking stationery* on page 4-10.)

   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 checkbox option can be used to provide more information about the output produced:

   Echo ARM tool command-lines verbosely

   To display the build commands in the Errors & Warnings window, select this checkbox.

   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.

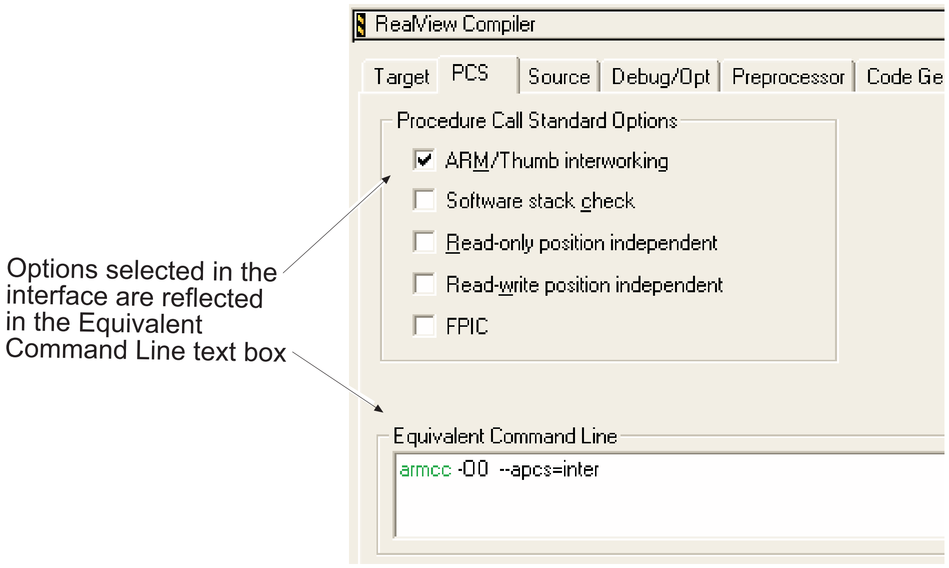
6. Click **Apply** to apply your changes.

10.4 Using the Equivalent Command Line text box

The Equivalent Command Line text box is displayed at the bottom of configuration panels for each of the:

- RealView Language Settings panels
- RealView Linker panels
- RealView Target panels
- RealView Debugger Panels
- Miscellaneous panels

Figure 10-13 shows an example for the RealView compiler configuration panel.


RealView Compiler
Target
PCS
Source
Debug/Opt
Preprocessor
Code Ge
Options selected in the interface are reflected in the Equivalent Command Line text box

Procedure Call Standard Options

- [✓] ARM/Thumb interworking
- [ ] Software stack check
- [ ] Read-only position independent
- [ ] Read-write position independent
- [ ] FPIC

Equivalent Command Line

```
armcc -O0 --apcs=inter
```

Figure 10-13 Equivalent Command Line text box

The Equivalent Command Line text box:

- Displays the command-line equivalent for any tool options you select in the panel. For example, if you select the **ARM/Thumb interworking** option in the RealView Compiler PCS panel, the Equivalent Command Line text box changes to display `--apcs=inter` on the command line.

- Enables you to edit the command line, and enter command-line options for which there are 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. If you want to use a tool command-line option from the CodeWarrior IDE, you can enter it in the text box. See *RealView Compilation Tools Linker and Utilities Guide* for more information on the command-line options to the ARM RealView tools.

- 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. If an option has an equivalent panel interface setting then this setting will be updated on the panel. Any option which has been entered on the command line that is not recognised will be highlighted in red.

10.5 Configuring assembler and compiler language settings

This section describes the language settings group of panels. These panels provide configuration options that are specific to the ARM RealView compiler and assembler:

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 10-36 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 options
- language mode, for example ANSI C or strict ANSI C
- debug and optimization options
- warning, error, and other preferences.

See *Configuring the RealView compiler* on page 10-47 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 tool chain see:

- *Configuring linker settings* on page 10-64
- *Configuring the debugger* on page 10-83.

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. Where necessary, you are referred to more detailed descriptions in the *RealView Compilation Tools Assembler Guide*.

Note

Many of the configuration settings described here use sensible default settings. However, you should review at least the Target options and PCS options to ensure that they are suitable for your project.

The configuration options are described in:

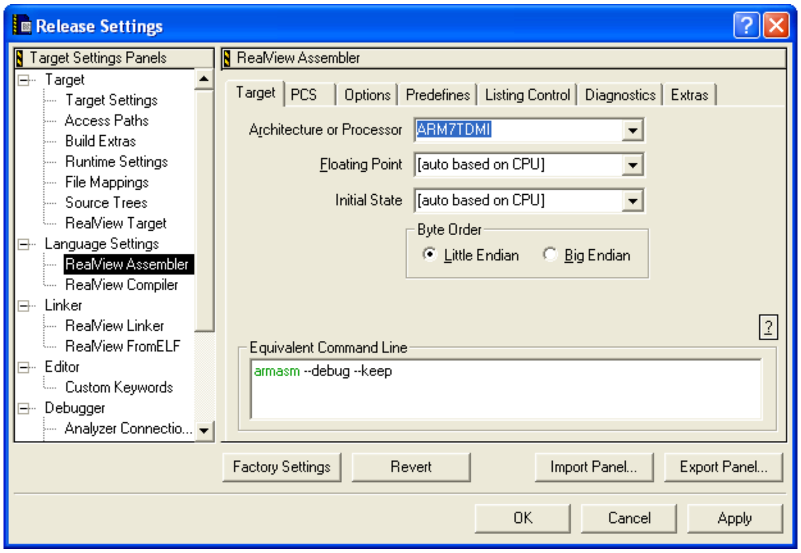
- *Configuring the target*
- *Configuring assembler PCS options* on page 10-39
- *Configuring miscellaneous assembler options* on page 10-41
- *Configuring predefined variables* on page 10-42
- *Configuring code listings* on page 10-44
- *Configuring diagnostic information* on page 10-45
- *Reading assembler options from a file* on page 10-46.

These sections provide general descriptions of the available assembler options. Where necessary, you are referred to more detailed descriptions in the *RealView Compilation Tools Assembler Guide*.

Configuring the target

Use the Target panel to configure the target processor and architecture for the RealView assembler:

1. Display the Target Settings 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 (Figure 10-14 on page 10-37).



Architecture or Processor: ARM7TDMI

Floating Point: [auto based on CPU]

Initial State: [auto based on CPU]

Byte Order: [✓ Little Endian] Big Endian

Equivalent Command Line

```
armasm --debug --keep
```

Figure 10-14 RealView Assembler Target panel

3. Select values for the following options:

Architecture or Processor

Select the processor or architecture for your target hardware from the drop-down list. Some processor-specific instructions produce either errors or warnings if assembled for the wrong target architecture.

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.

Valid values are:

[auto based on CPU]

Use the floating-point selection based on the **Architecture or Processor** selection. This is the default setting.

VFPv2

Select this option to target hardware vector floating-point units conforming to architecture VFPv2, such as the ARM1176JZF-S.

SoftVFP

Select this option to use the software floating-point library (FPlib).

SoftVFP+VFPv2

Selects software floating-point library with pure-endian doubles, software floating-point linkage, and requiring 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.

None

Select this option if you are neither targeting a hardware floating-point unit, nor the software floating-point library. This produces an error if your code contains floats.

Initial State

This option sets up the initial state of the assembler to assemble instructions in the source as ARM or Thumb instructions. This state will remain until an ARM, THUMB, CODE16 or CODE32 directive is processed in the code to change the state of the assembler.

Valid values are:

auto based on CPU

Uses the default state for the selected processor. This is the default setting.

ARM

Start the assembler in ARM state. This instructs the assembler to interpret instructions as ARM instructions. This is equivalent to `--arm` or `--32` options.

Thumb

Start the assembler in Thumb state. This instructs the assembler to interpret instructions as Thumb instructions. This is equivalent to the `--thumb` option.

Legacy Thumb

Start the assembler in Thumb state and accept instructions in 'Legacy' Thumb syntax. This instructs the assembler to interpret instructions as Thumb instructions using the old Thumb syntax. This is equivalent to the `--16` option.

Byte Order

Select the byte order used by your target hardware. This can be either **Little Endian** or **Big Endian**. The default setting is **Little Endian**.

4. Click **Apply** to apply your changes.

Configuring assembler 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** ———

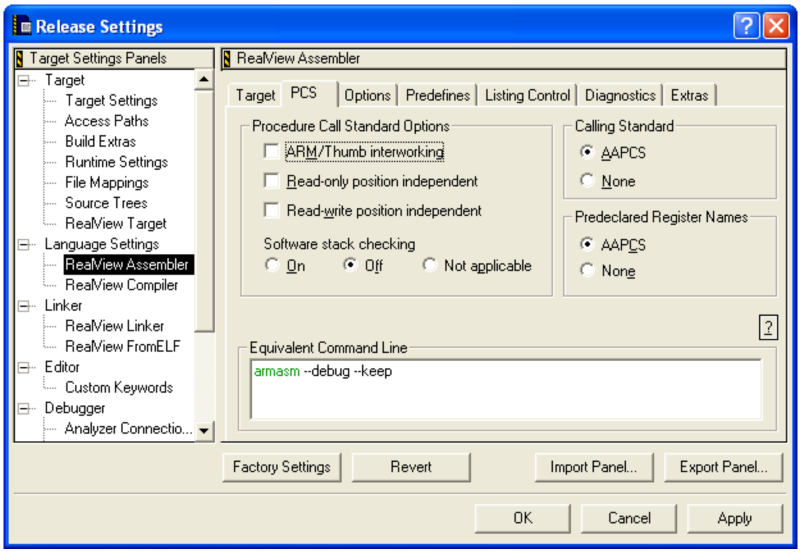
Selecting PCS options does *not* provide checks that your assembly language code conforms to a given PCS variant. You must ensure that your assembly language code follows the conventions required by the PCS options you select. See the appendix on using the PCS in the *RealView Compilation Tools Developer Guide* for more information.

———————————

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 compiler PCS options* on page 10-51 for details.

To configure the PCS settings for the RealView assembler:

1. Display the Target Settings 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 **PCS** tab to display the configuration panel (Figure 10-15).



Procedure Call Standard Options
- [ ] ARM/Thumb interworking
- [ ] Read-only position independent
- [ ] Read-write position independent

Software stack checking: On [✓ Off] Not applicable

Calling Standard: [✓ AAPCS] None

Predeclared Register Names: [✓ AAPCS] None

Equivalent Command Line

```
armasm --debug --keep
```

Figure 10-15 RealView Assembler PCS panel

3. Select values for the calling standard and predefined register names options:

   Calling Standard

   Select **AAPCS** (ARM Architecture Procedure Calling Standard) if your assembly language code is expected to conform to one of the PCS variants. You should select **AAPCS** if you are writing assembly language routines that are called from C or C++, or from other assembly language routines that expect your code to follow PCS calling conventions.

   You can select **None** if you are writing standalone assembly language routines that are not called from other routines that expect PCS calling conventions to be followed. In this case you are responsible for maintaining your own register usage conventions.

   Predeclared Register Names

   Select AAPCS if you are writing assembly language routines that conform to one of the AAPCS variants and you want the assembler to recognize the conventional predefined register names, such as `a1-a4`. See the *RealView Compilation Tools Assembler Guide* for a complete list of predefined register names.

4. Select the Procedure Call Standard variant options that you require. The following options are available in 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 output by the assembler.

   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 output by the assembler.

Software stack checking

Select the software stack checking option you require:

On Select this option if you are writing code that performs stack checking in software.

Off Select this option if you are writing code for a system that does not require software stack checking.

Not Applicable

Select this option if you are writing code that can work with either software stack checking, or non software stack checking code.

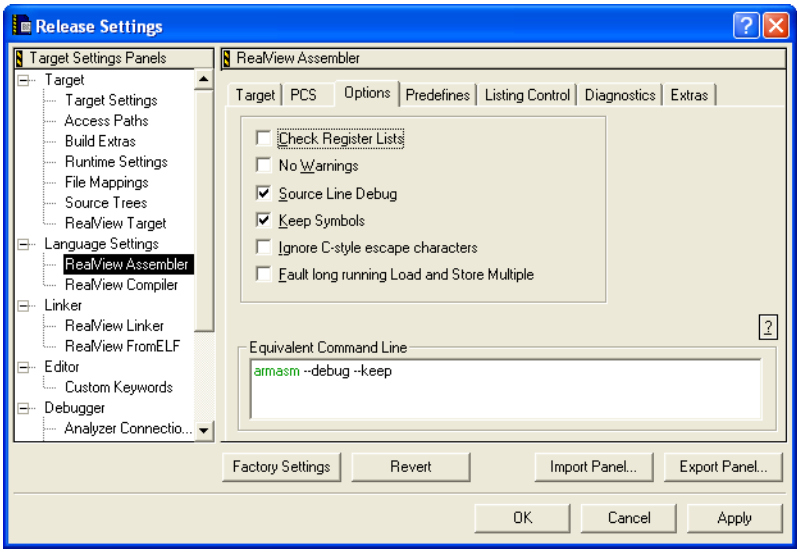
5. Click **Apply** to apply your settings.

Configuring miscellaneous assembler options

To configure other miscellaneous assembler options, such as code checks, warnings, and debug options:

1. Display the Target Settings 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).



- [ ] Check Register Lists
- [ ] No Warnings
- [x] Source Line Debug
- [x] Keep Symbols
- [ ] Ignore C-style escape characters
- [ ] Fault long running Load and Store Multiple

Equivalent Command Line

```
armasm --debug --keep
```

Figure 10-16 RealView Assembler Options panel

3. Select values for the following options:

   Check Register Lists

   Select this option to instruct the assembler to check `RLIST`, `LDM`, and `STM` register lists to ensure that all registers are provided in increasing register number order. If this is not the case, a warning is given.

   No Warnings

   Select this option to turn off all warning messages.

   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.

   Ignore C-style escape characters

   Select this option to instruct the assembler to ignore C-style escaped special characters, such as `\n` and `\t`.

   Fault long running Load and Store Multiple

   Select this option to instruct the assembler to fault LDM and STM instructions if the maximum number of registers transferred exceeds:

   - five, for all STMs, and for LDMs that do not load the PC
   - four, for LDMs that load the PC.

   See the *RealView Compilation Tools Assembler Guide* for detailed information on using this option.

4. Click **Apply** to apply your settings.

Configuring predefined variables

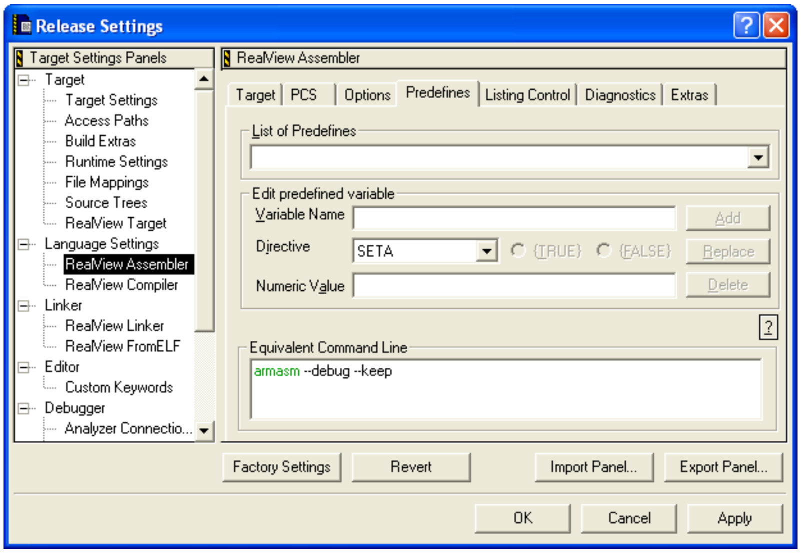
Use the Predefines panel to configure the assembler to predefine a global variable and execute one of the `SET` directives to set its value, for example:

```
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 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).



Directive: SETA

```
armasm --debug --keep
```

Figure 10-17 RealView Assembler Predefines 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 list.
4. Select the directive you require. Use:
   - `SETA`, for numeric values
   - `SETS`, for string values
   - `SETL`, for logical values.
5. Enter the value of the variable in the value field, or click the appropriate boolean value button if you have selected a `SETL` directive.

   ——— **Note** ———

   If you want use the `SETS` directive to specify a predefined variable, you can use a quoted string in the String Value field, for example, "My string". When you add the variable, CodeWarrior automatically inserts the escape character prefix \ for the quotes, for example

```
--predefine="STRING_DEF SETS \"My string\""
```

However, if you want to add this line to the Equivalent Command Line field directly, you must include the escape characters yourself.

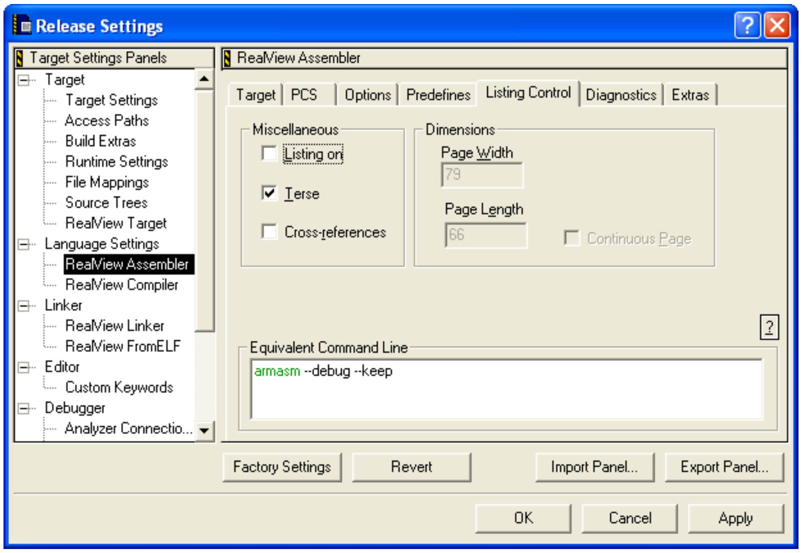
---

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.

Configuring code listings

To configure options for code listings 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 **Listing Control** tab to display the configuration panel (Figure 10-18).



- [ ] Listing on
- [✓] Terse
- [ ] Cross-references

Page Width: 79

Page Length: 66

- [ ] Continuous Page

```
armasm --debug --keep
```

Figure 10-18 RealView Assembler Listing Control panel

3. Select values for the following options, as required:

   Listing on
   : Select this option to instruct the assembler to output a detailed listing of the assembly language produced by the assembler after it has resolved assembler constructs such as directives and pseudo-instructions. The listing is displayed in a new text window. You can edit how the listing is displayed in this window from 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.

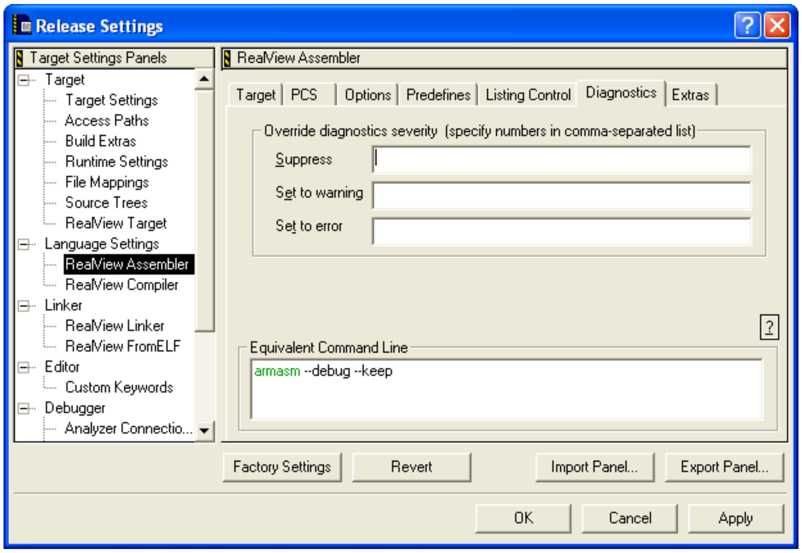
   Dimensions
   : Use the Page Width and Page Length fields to set the page width and page length for your listings. Select the **Continuous Page** option if you do not want page breaks inserted in the listing.

4. Click **Apply** to apply your settings.

Configuring diagnostic information

To control the output of the diagnostic error 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 **Diagnostic** tab to display the configuration panel (Figure 10-18 on page 10-44).



```
armasm --debug --keep
```

Figure 10-19 RealView Assembler Diagnostics panel

3. Select values for the following options, as required:

   Suppress
   Disables all diagnostic messages being produced for warnings with the specified tags. Enter a list of tags separated by commas.

   Set to warning
   Sets the diagnostic messages that have the specified tags to the warning severity.

   Set to error
   Sets the diagnostic messages that have the specified tags to the error severity.

4. Click **Apply** to apply your settings.

Reading assembler options from a file

Use the Extras 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. You can use via files:

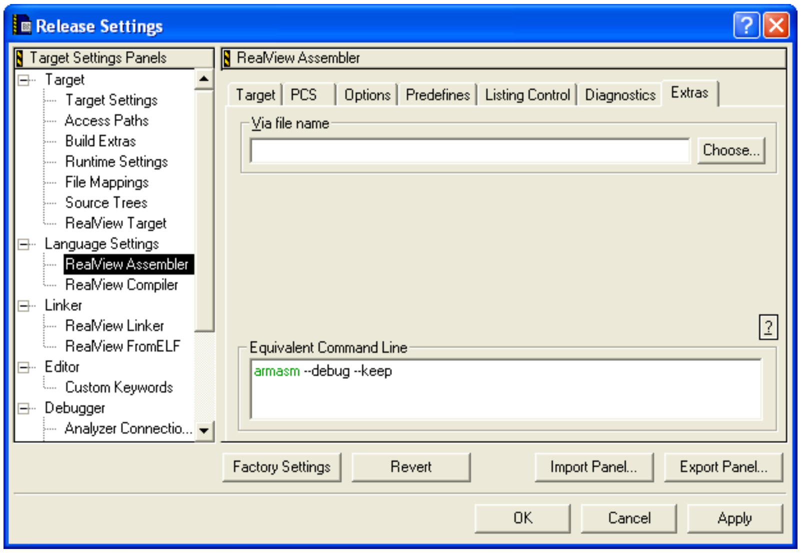
- to ensure that the same assembler settings are used by different build targets and projects
- to include exceptionally long command line options to the assembler

See the *RealView Compilation Tools v2.2 Compiler and Libraries 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 10-20).



```
armasm --debug --keep
```

Figure 10-20 RealView Assembler Extras panel

3. Enter the path name of the via file, or click **Choose…** and select the via file from the standard file dialog. The via file option is placed at the end of the command line and will therefore override any options specified in the other assembler configuration panels if there is a conflict.

4. Click **Apply** to apply your settings.

10.5.2 Configuring the RealView compiler

This section describes how to configure the RealView compiler (`armcc`) used for compiling C and C++ source files.

Note

Many of the configuration settings described here use sensible default settings. However, you should review at least the Target options and PCS options to ensure that they are suitable for your project.

The compiler panel consists of a number of tabbed panes. The panels are listed in the Target Settings Panels list (see Figure 10-21 on page 10-49 for an example) when you select the ARM RealView linker or the ARM RealView Librarian as the linker in the Target Settings panel. See *Configuring target settings* on page 10-8 for more information on selecting the linker in the Target Settings panel.

The configuration options are described in:

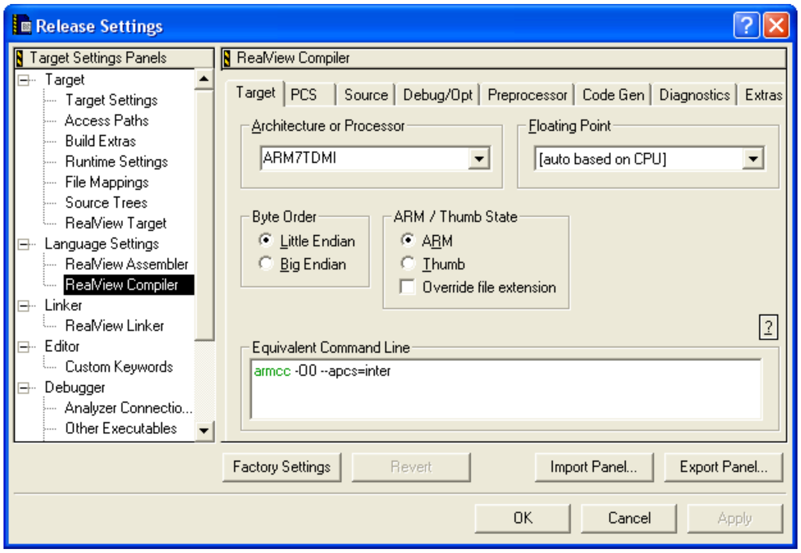
- *Configuring the target and architecture*
- *Configuring compiler PCS options* on page 10-51
- *Configuring sources* on page 10-53
- *Configuring debug and optimization* on page 10-55
- *Configuring the preprocessor* on page 10-58
- *Configuring code generation* on page 10-59.
- *Configuring diagnostic information* on page 10-61
- *Reading compiler options from a file* on page 10-62.

These sections give general descriptions of the available compiler options. Where applicable, you are referred to more detailed descriptions in the C and C++ Compiler chapters of the *RealView Compilation Tools v2.2 Compiler and Libraries Guide*.

Configuring the target and architecture

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 10-21 on page 10-49).



Architecture or Processor: ARM7TDMI

Floating Point: [auto based on CPU]

Byte Order: (●) Little Endian ( ) Big Endian

ARM / Thumb State: (●) ARM ( ) Thumb [ ] Override file extension

Equivalent Command Line:

```
armcc -O0 --apcs=inter
```

Figure 10-21 RealView compiler Target and Source panel

3. Select values for the following options:

Architecture or Processor

Select an architecture or processor for your target hardware from the drop-down list. The drop-down list contains menu items for all current product names and architectures. If you select:

- an architecture, for example 4T, your code is compiled to run on any processor that implements that architecture
- a processor, the compiler compiles code that is optimized for that processor.

Some processor selections imply a floating-point selection. For example, with the RealView compiler ARM1176JZF-S implies VFPv2 format and instruction. The implied option is overridden if you specify an explicit floating-point option.

Floating Point

Select the floating-point system you are using. Valid values are:

[auto based on CPU]

Use the floating-point selection based on the **Architecture or Processor** selection. This is the default setting.

VFPv2

Select this option to target hardware vector floating-point units conforming to architecture VFPv2, such as the ARM1176JZF-S.

SoftVFP

Select this option to use the software floating-point library with pure-endian doubles (FPlib).

SoftVFP+VFPv2

Selects software floating-point library with pure-endian doubles, software floating-point linkage, and requiring 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.

None

Select this option if you are neither targeting a hardware floating-point unit, nor the software floating-point library. This produces an error if your code contains floats.

Byte Order

Select the byte order used by your target hardware.

ARM / Thumb State

To configure the compiler for handling C and C++ source files in a target, edit the **ARM / Thumb State** group box:

ARM

Compile all C and C++ files as ARM instructions but make adjustments to compile files with `.tc` and `.tcpp` filename extensions as Thumb instructions.

Thumb

Compile all C and C++ sources as Thumb instructions but make adjustments to compile `.ac` and `.acpp` filename extensions as ARM instructions.

Override file extension

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 either ARM instructions or Thumb instructions. For example, if this option is set with ARM state, the RealView compiler will *not* switch to compile `.tc`, `.tcpp` files as Thumb instructions. It will compile these files as ARM instructions.

4. Click **Apply** to apply your changes.

Configuring compiler PCS options

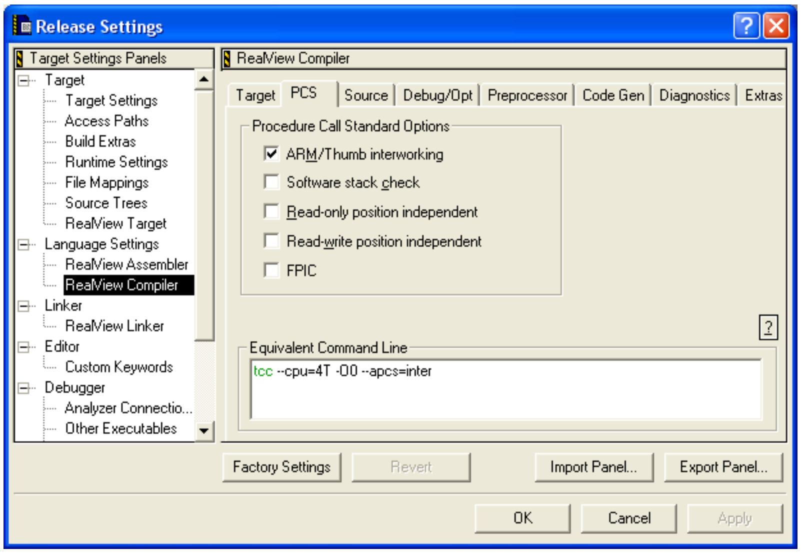
Selecting the PCS (Procedure Call Standard) options instructs the compiler to generate code that conforms to the appropriate PCS variant. In general, you must ensure that you use compatible calling standard options when you are compiling objects or libraries that you expect to link together.

In addition, if you are calling routines written in ARM assembly language you must ensure that your assembly language code conforms to the appropriate PCS variant, and that the assembler is configured with the appropriate PCS options. See *Configuring assembler PCS options* on page 10-39 for details. See also the PCS chapter in the *RealView Compilation Tools Developer Guide* for more information.

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** tab to display the configuration panel (Figure 10-22).



Procedure Call Standard Options

- [✓] ARM/Thumb interworking
- [ ] Software stack check
- [ ] Read-only position independent
- [ ] Read-write position independent
- [ ] FPIC

Equivalent Command Line

```
tcc --cpu=4T -O0 --apcs=inter
```

Figure 10-22 RealView compiler PCS panel

3. Select the PCS variant options that you require:

   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

   We recommend 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.

   Software stack check

   Select this option if your code requires stack checking in software.

   Read-only position independent

   Select this option if you are compiling code that you want to be position-independent. (See the `ROPI` option in the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for more details about this option.)

   Read-write position independent

   Select this option to ensure that output data sections in your compiled code are addressed position-independently. (See the `RWPI` option in the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for more details about this option.)

   FPIC

   Selecting this option configures the compiler to generate read-only position-independent code where relative address references are independent of the location where your program is loaded. Relative addressing is only implemented when your code makes use of System V shared libraries. If your code uses shared objects, you do not have to compile with this option. (See the `FPIC` option in the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for more details about this option.)

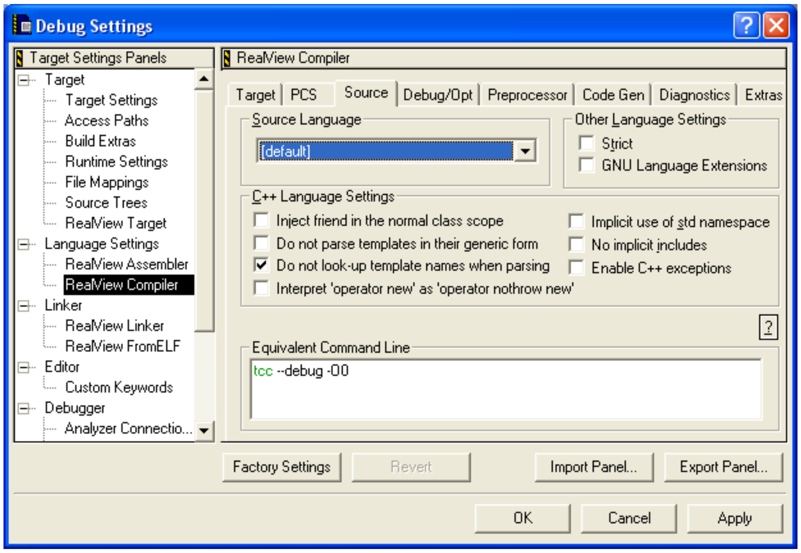
4. Click **Apply** to apply your changes.

Configuring sources

Selecting the Source options instructs the compiler about the source language you are using and how you wish it to be compiled. See the compiler chapters of the *RealView Compilation Tools v2.2 Compiler and Libraries Guide*for detailed information on C and C++ standards conformance and implementation details.

To configure the 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 10-27 on page 10-61).



Source Language: [default]

Other Language Settings:
- ☐ Strict
- ☐ GNU Language Extensions

C++ Language Settings:
- ☐ Inject friend in the normal class scope
- ☐ Do not parse templates in their generic form
- ☑ Do not look-up template names when parsing
- ☐ Interpret 'operator new' as 'operator nothrow new'
- ☐ Implicit use of std namespace
- ☐ No implicit includes
- ☐ Enable C++ exceptions

Equivalent Command Line:

```
tcc --debug -O0
```

Figure 10-23 RealView compiler Source panel

3. Choose the language used by the compiler from the Source Language drop-down list box:

[default]

Select this option to compile C and C++ sources using the default languages used by the compiler:

- ISO Standard C (1990); for C code
- ISO Standard C++ (1998); for C++ code.

You can choose to compile all C and C++ sources using just one of these standards by choosing the appropriate option from the drop-down list box.

ISO Standard C (1990)

Select this option to enable compilation of *all* C and C++ sources in the target as C, specifically the ISO Standard C (1990) version of C. (This is the language used by the compiler for all C files when the **[default]** Source Language option is selected.)

ISO Standard C (1990) is a fairly strict ISO compiler, but without some of the inconvenient features of the ISO C Standard. Some minor extensions are also supported, for example `//` in comments and `$` in identifiers.

ISO Standard C++ (1998)

Select this option to enable compilation of *all* C and C++ sources in the target as C++, specifically the ISO Standard C++ (1998). (This is the language used by the compiler for all C++ files when the **[default]** Source Language option is selected.)

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. Refer to the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for more information.

GNU Language Extensions

Enables or disables the GNU compiler extensions that are supported by the compiler. Refer to the *RealView Compilation Tools v2.2 Compiler and Libraries Guide* for more information.

5. Select any of the following 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 actually instantiated. If dependent name processing is enabled, then parsing is done 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.

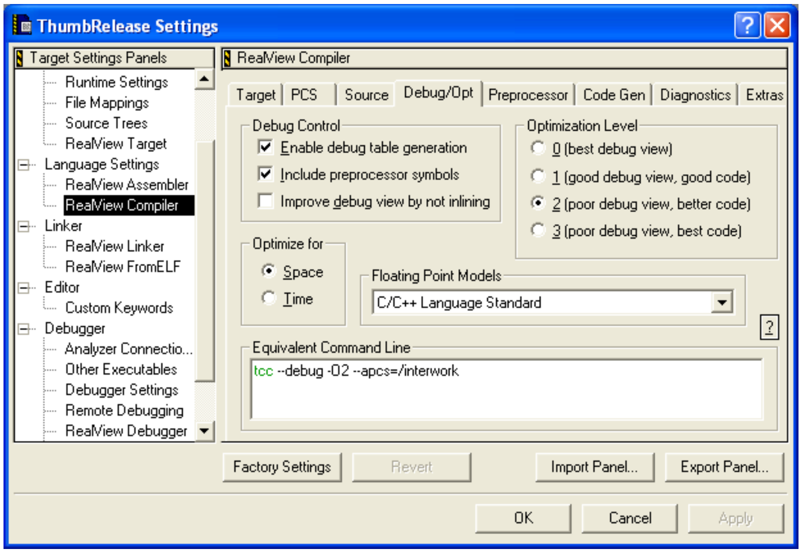
6. Click **Apply** to apply your changes.

Configuring debug and optimization

Use the **Debug/Opt** 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 10-24 on page 10-56).



Debug Control
- [✓] Enable debug table generation
- [✓] Include preprocessor symbols
- [ ] Improve debug view by not inlining

Optimization Level
- 0 (best debug view)
- 1 (good debug view, good code)
- [✓ 2 (poor debug view, better code)]
- 3 (poor debug view, best code)

Optimize for
- [✓ Space]
- Time

Floating Point Models: C/C++ Language Standard

Equivalent Command Line

```
tcc --debug -O2 --apcs=/interwork
```

Figure 10-24 RealView Compiler Debug and optimization 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 will not be available for the resultant image.

Include preprocessor symbols

Select this option to include preprocessor symbols in the compiler output.

Improve debug view by not inlining

Select this option to instruct the compiler to compile `inline` qualified functions out of line so that they can be debugged at source more easily.

4. Select the level of optimization you want:

0 (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 (good debug view, good code)

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 (poor debug view, better code)

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 (poor debug view, best code)

Select this option to enable all compiler optimizations. This option may result in a poor debug view of your output image due to code movement and register re-use.

5. Select the floating-point model to use. The model specifies floating-point conformance and sets library attributes and floating-point optimizations:

   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 will behave correctly.

6. Select the optimization criterion:

For space	Select this option to favor small code size over execution speed. This is the default.
For time	Select this option to favor execution speed over code size.

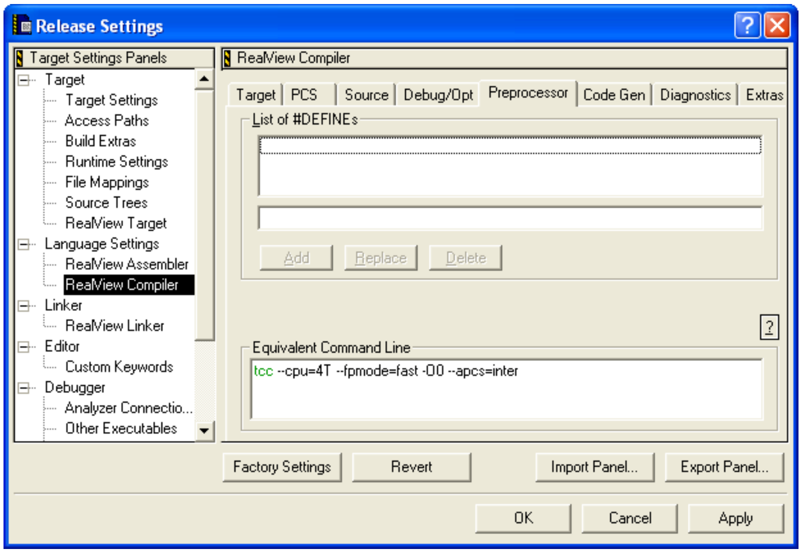
7. Click **Apply** to apply your changes.

Configuring the preprocessor

Use the Preprocessor 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).

2. Click RealView Compiler in the Target Settings Panels list and click the **Preprocessor** tab to display the configuration panel (Figure 10-25). The panel displays a list of all predefined macros.



```
tcc --cpu=4T --fpmode=fast -O0 --apcs=inter
```

Figure 10-25 RealView Compiler Preprocessor panel

3. Double-click on a preprocessor macro definition in the **List of #DEFINEs** box to modify the definition. If you want to add a new definition, type the macro name in the edit field (below the **List of #DEFINEs** box) and click on **Add**. The macro name is added to the **List of #DEFINEs** box.

4. Edit the value of the macro definition as appropriate. If you want to create a new macro, type over the existing macro name. Specify the value of the macro with an equals sign. For example:

   ```
   EXAMPLE_DEFINE=2
   ```

   If you do not enter a value, the value defaults to 1.

5. Click **Add** to add a new macro definition, or click **Replace** to edit the value of an existing macro definition.

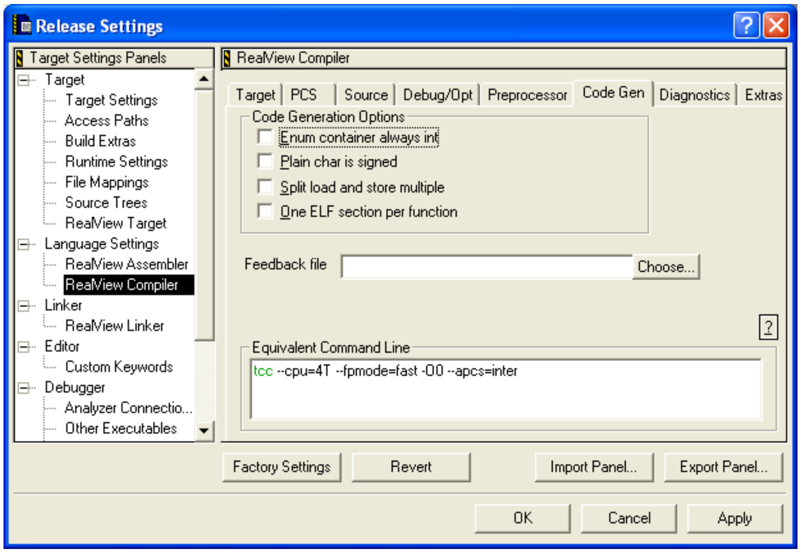
6. Click **Apply** to apply your changes.

Configuring code generation

To configure code generation options 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 **Code Gen** tab to display the configuration panel (Figure 10-26).



Code Generation Options
- [ ] Enum container always int
- [ ] Plain char is signed
- [ ] Split load and store multiple
- [ ] One ELF section per function

Feedback file [ ] Choose...

Equivalent Command Line

```
tcc --cpu=4T --fpmode=fast -O0 --apcs=inter
```

Figure 10-26 RealView Compiler Code Gen panel

3. Select Code Generation options:

   Enum container always int

   Select this option to force all enumerations to be stored in integers. By default, the compiler uses the smallest data type that can hold all values in an `enum`.

   Note

   This option is not recommended for general use and is not required for ISO-compatible source. Code compiled with this option is not compliant with the ABI for the ARM Architecture (base standard) [BSABI], and incorrect use might result in a failure at runtime. This option is not supported by the C++ libraries.

   Plain char is signed

   Select this option to make the `char` type signed. It is normally unsigned in C++ and ANSI C modes.

   Note

   This option is not recommended for general use and is not required for ISO-compatible source. Code compiled with this option is not compliant with the ABI for the ARM Architecture (base standard) [BSABI], and incorrect use might result in a failure at runtime. This option is not supported by the C++ libraries.

   Split load and store multiple

   Select this option to instruct the compiler to split `LDM` and `STM` instructions into two or more `LDM` or `STM` instructions, where required, to reduce the maximum number of registers transferred to:

   - five, for all `STM`s, and for `LDM`s that do not load the PC
   - four, for `LDM`s that load the PC.

   One ELF section per function

   Select this option 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.

4. 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.

5. Click **Apply** to apply your changes.

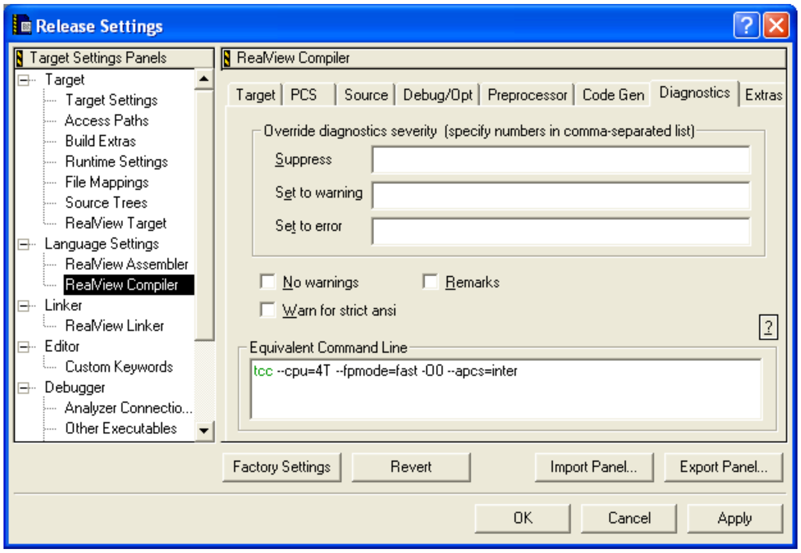
Configuring diagnostic information

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 configure warnings given by 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 **Diagnostics** tab to display the configuration panel (Figure 10-27).



```
tcc --cpu=4T --fpmode=fast -O0 --apcs=inter
```

Figure 10-27 RealView compiler Diagnostics panel

3. In the **Override diagnostics severity** group box, select values for the following options, as required:

   Suppress

   Disable diagnostic messages being produced for warnings with the specified tags. Enter a list of tags separated by commas.

   Set to warning

   Sets the diagnostic messages that have the specified tags to the warning severity.

Set to error

Sets the diagnostic messages that have the specified tags to the error severity.

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 that are issued when extensions to the ANSI standard are used implicitly. Examples include:

   - 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.

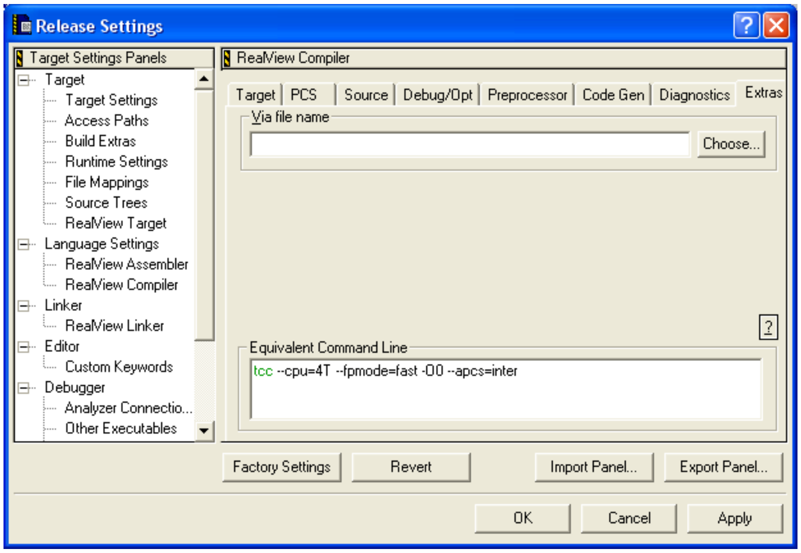
Reading compiler options from a file

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. See the *RealView Compilation Tools v2.2 Compiler and Libraries 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. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 10-28 on page 10-63).



```
tcc --cpu=4T --fpmode=fast -O0 --apcs=inter
```

Figure 10-28 RealView Compiler Extras 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.

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 the linker that 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 subprojects or subtargets to select additional linkers or post-linkers. See *Working with multiple build targets and subprojects* on page 3-44 for more information.

You can use the following linker and post-linker options with the ARM tool chain:

RealView librarian

There is no configuration panel required 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 4-17 for more information. See also the utilities chapter of the *RealView Compilation Tools Linker and Utilities Guide*.

RealView Linker

Use this panel to configure the RealView linker. See *Configuring the RealView linker* on page 10-65 for details.

RealView FromELF

Use this panel 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 10-75 for details.

The Batch File Runner

There is no configuration panel required for the batch runner. See *Running batch files with the batch runner* on page 4-51 for more information.

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.

10.6.1 Configuring the RealView linker

This section describes how to configure options for the RealView linker (`armlink`). It provides a general description of the options that you can configure through the CodeWarrior IDE. See the *RealView Compilation Tools Linker and Utilities Guide* for a description of:

- the RealView linker and its command-line options
- how the linker constructs images and partially linked objects
- scatter-loading.

Linker configuration options are described in:

- *Configuring linker output*
- *Configuring linker options* on page 10-68
- *Configuring image layout* on page 10-70
- *Configuring linker listings* on page 10-72
- *Configuring linker diagnostics* on page 10-73
- *Configuring linker extras* on page 10-74.

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:

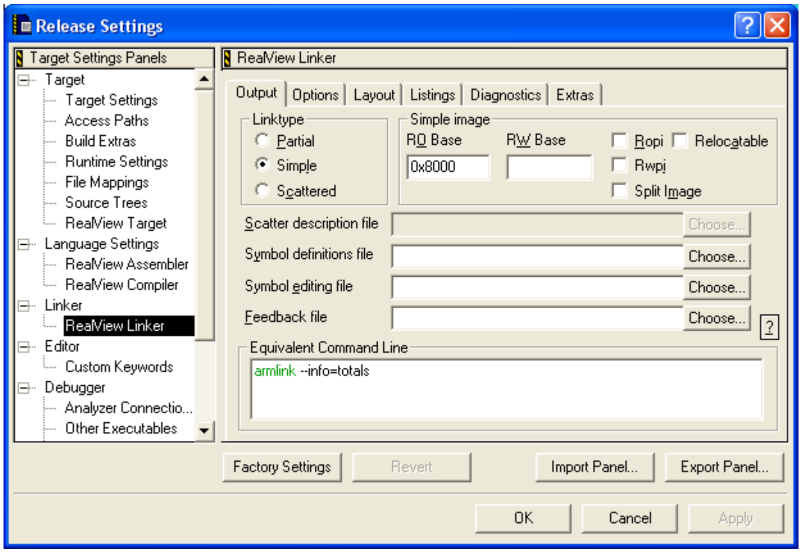
- A partially linked object. You can use this option to produce a partially linked ELF object file that you can use in a later linker or Librarian operation. See also *Working with multiple build targets and subprojects* on page 3-44 for information on configuring dependent subtargets and subprojects.
- A simple image that does not require a scatter-load description file to describe the structure of the image. This option provides an easy way to produce an executable ELF image. It gives limited control over the structure of the image.
- A scatter-loaded image. Use this output type for more detailed control over the linker output. You must write a scatter-load description file for the image you want to produce.

See the *RealView Compilation Tools Linker and Utilities Guide* for detailed information on how to control output from the linker, including a description of scatter-loading. See also the description of writing code for ROM in the *RealView Compilation Tools Developer Guide* for guidance on producing images suitable for embedding in ROM.

To set the output options for images:

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 **Output** tab to display the configuration panel (Figure 10-29).



Linktype: ( ) Partial [✓ Simple] ( ) Scattered

RO Base: 0x8000

```
armlink --info=totals
```

Figure 10-29 RealView linker Output panel

3. Select the type of image you want to produce in the **Linktype** group of options. Select one of:

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-load description file. If you select this option the Simple image group of options becomes active and a Layout panel tab is added to the list of available panels. See *Configuring image layout* on page 10-70 for more information on specifying layout options when linking a simple image. See below for more information on setting the simple image options.

Scattered

Select this option if you want to use a scatter-load 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 pathname of the file in the **Scatter description file** text field
- click **Choose…** to select the file from the standard file dialog box.

By convention, scatter-load description files use a filename extension of `.scat`. You can add scatter-load description files to your project. See the *RealView Compilation Tools Linker and Utilities Guide* for information on writing a scatter-load description file.

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 load 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, that contains the RO and RW output sections, into the following load regions:

- One 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 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, **RW Base** 0 is assumed.

Both regions are root regions.

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 symbol definitions file, if required. See the description of the `--symbols` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information on symbol files.

6. Select a symbol editing file, if required. See the description of the `--edit` option in the *RealView Compilation Tools Linker and Utilities Guide* for more information on editing symbols in output objects.

7. Select a feedback file, if required. The file is used to keep a record of unused functions. It is used by the compiler. See the description of the `--feedback` option in the *RealView Compilation Tools Linker and Utilities Guide*

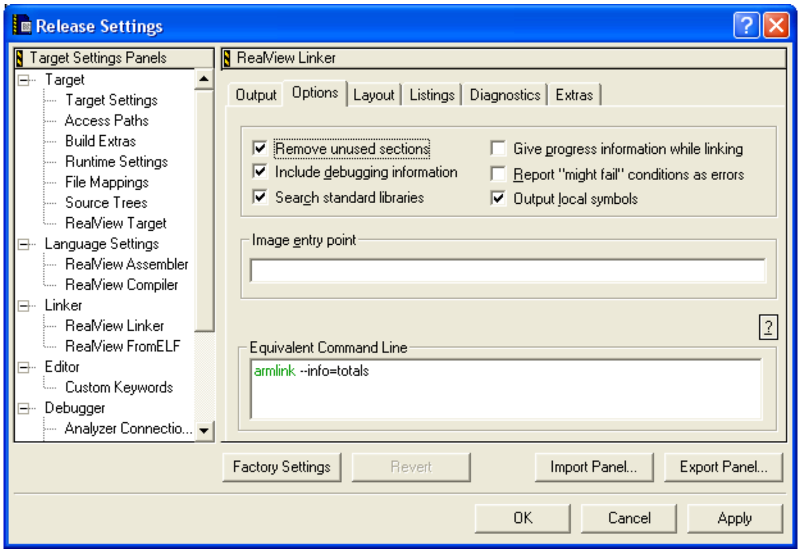
8. Click **Apply** to apply your changes.

Configuring linker options

Use the linker options panel to configure options such as unused section elimination and symbol resolution. To configure 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 **Options** tab to display the configuration panel (Figure 10-30 on page 10-69).



- [✓] Remove unused sections
- [✓] Include debugging information
- [✓] Search standard libraries
- [ ] Give progress information while linking
- [ ] Report "might fail" conditions as errors
- [✓] Output local symbols

Image entry point

Equivalent Command Line

```
armlink --info=totals
```

Figure 10-30 RealView Linker Options panel

3. Select additional linker options, as required. The following options are available:

Remove unused sections

Select this option to remove unused sections from the image. This retains all input sections in the final image even if they are unused.

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.

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 a link.

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. You can specify an entry point in the following forms:
   - as a unique entry point address
   - as the start of the input section that defines a symbol
   - as an offset inside a section within a specific object.

   The entry point specified here is used to set the ELF image entry address. See the description of the `--entry` linker option in the linker chapter of the *RealView Compilation Tools Linker and Utilities Guide* for details.

5. Click **Apply** to apply your changes.

Configuring image layout

You can use the **Image Layout** panel to specify which sections should be placed first and last in an image:

- The **Place at Beginning of Image** options enable you to place a selected input section first in its execution region.
- The **Place at End of Image** options enable you to place a selected input section last in its execution region.

You can use this, 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.

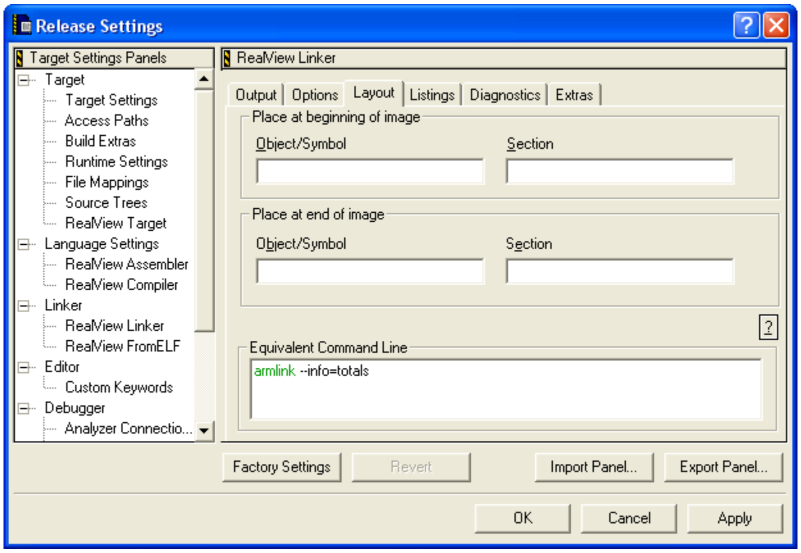
Note

You can use scatter-loading 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 **Image Layout** tab to display the configuration panel (Figure 10-31).



```
armlink --info=totals
```

Figure 10-31 RealView Linker Layout panel

3. Enter section specifications in the appropriate fields, as required. You can enter specifiers for either, or both, the first and last sections. 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.
   - An object name, and section label. This selects the specified section from within the specified object.

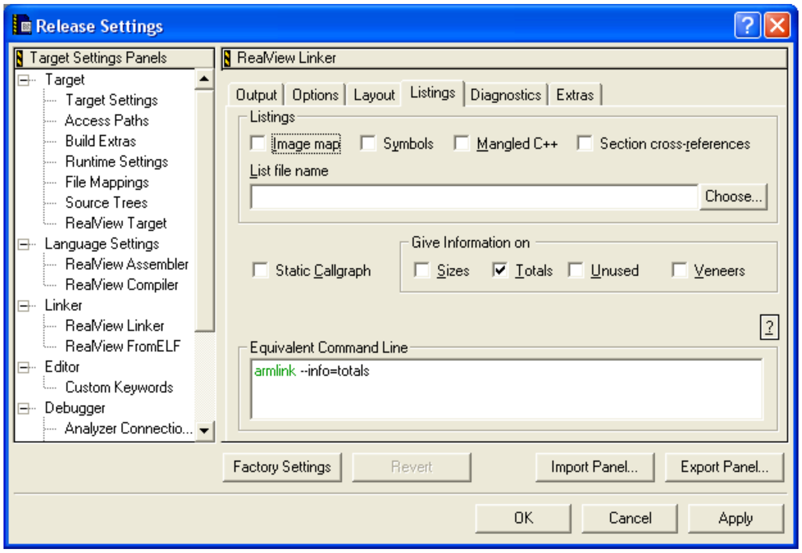
   See the *RealView Compilation Tools Linker and Utilities Guide* for detailed information on specifying first and last sections.

4. Click **Apply** to apply your changes.

Configuring linker listings

Use the **Listings** panel to instruct the linker to output link information to a listing file. To configure how the linker produces listing information:

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 10-32).



Listings
- [ ] Image map
- [ ] Symbols
- [ ] Mangled C++
- [ ] Section cross-references

List file name

- [ ] Static Callgraph

Give Information on
- [ ] Sizes
- [x] Totals
- [ ] Unused
- [ ] Veneers

Equivalent Command Line

```
armlink --info=totals
```

Figure 10-32 RealView Linker Listings panel

3. Select the information you want to list during the link operation:

Image map

Select this option to create an image map listing the base and size of each region and section in the image.

Symbols Select this option to list each symbol used in the link step, including linker-generated symbols, and its value.

Mangled C++

Select this option to instruct the linker to display mangled C++ symbol names in diagnostic messages, and in listings produced by the various linker listing 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.

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. Select the **Static Callgraph** checkbox 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` linker option in the *RealView Compilation Tools Linker and Utilities Guide* for 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. See *Configuring linker options* on page 10-68 for more information on unused section elimination.

   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.

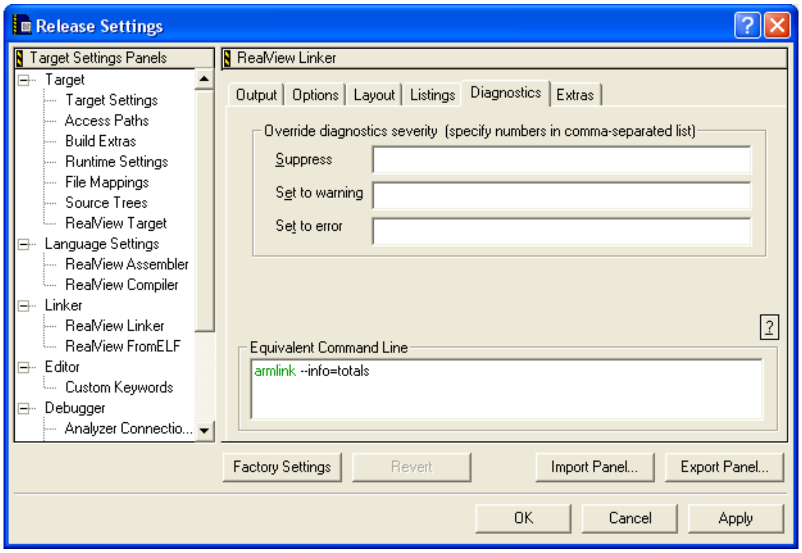
7. Click **Apply** to apply your changes.

Configuring linker diagnostics

Use the **Diagnostics** panel to configure extra 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 10-32 on page 10-72).



Override diagnostics severity (specify numbers in comma-separated list)

Suppress

Set to warning

Set to error

Equivalent Command Line

```
armlink --info=totals
```

Figure 10-33 RealView Linker Diagnostics panel

3. In the **Override diagnostics severity** group box, select values for the following options, as required:

   Suppress

   Disable diagnostic messages being produced for warnings with the specified tags. Enter a list of tags separated by commas.

   Set to warning

   Sets the diagnostic messages that have the specified tags to the warning severity.

   Set to error

   Sets the diagnostic messages that have the specified tags to the error severity.

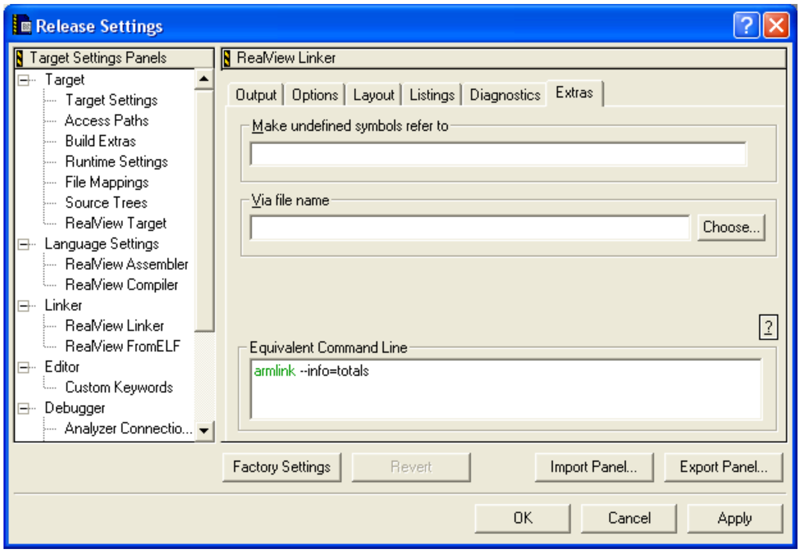
4. Click **Apply** to apply your settings.

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 on page 10-72).



```
armlink --info=totals
```

Figure 10-34 RealView Linker Extras panel

3. In the **Make undefined symbols refer to** edit field, 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 will fail. 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.

4. In the **Via file name** edit field, 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.

5. Click **Apply** to apply your settings.

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 `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 more information.

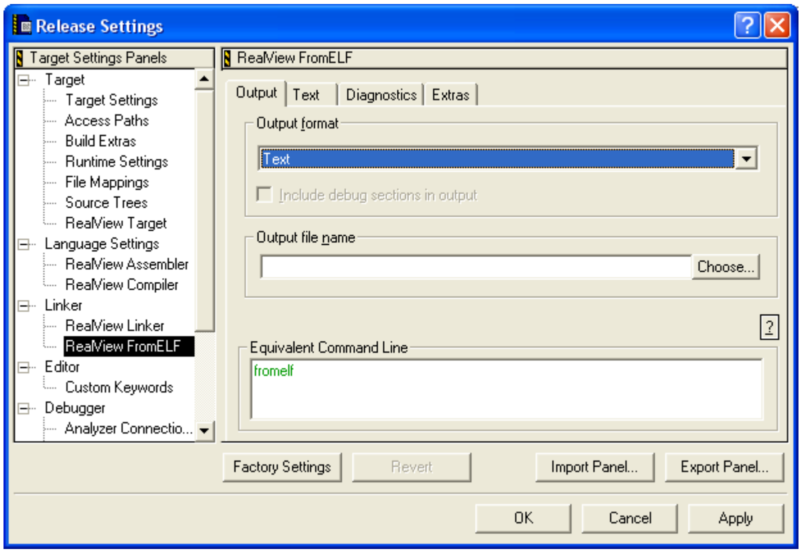
RealView `fromelf` configuration options are described in:

- *Configuring RealView fromelf output format*
- *Configuring RealView fromelf text format* on page 10-77
- *Configuring RealView fromelf diagnostics* on page 10-78
- *Reading RealView fromelf options from a file* on page 10-79

Configuring RealView fromelf 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 10-35).



Output format: Text

```
fromelf
```

Figure 10-35 RealView fromELF output panel

3. Click the **Output format** drop-down list and select the output format you want. Either:
   - Select from the binary output options. See the Toolkit Utilities chapter of the *RealView Compilation Tools Linker and Utilities Guide* for details.
   - Select **Text** to extract a text file of information on the output image. Selecting this option adds a Text panel to the **RealView fromELF** panel list from where you can select the text format (see *Configuring RealView fromelf text format* on page 10-77.

4. Select **Include debug sections in output** to ensure that debug table information is included in the `fromelf` output. (This option is only enabled if you choose **ELF** as your output format.)

5. Enter the pathname of the output file or click **Choose…** to select an output file from the standard file dialog box. If you do not enter a pathname:
   - output text information is displayed in a new editor window if the Text information output format is selected
   - the converted binary is saved in the target subdirectory of the project data directory if a binary output is selected.

6. Click **Apply** to apply your changes. When you make your project, the CodeWarrior IDE calls `fromelf` to process the output.

Configuring RealView fromelf text format

To configure the RealView `fromelf` text 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 10-35 on page 10-76).

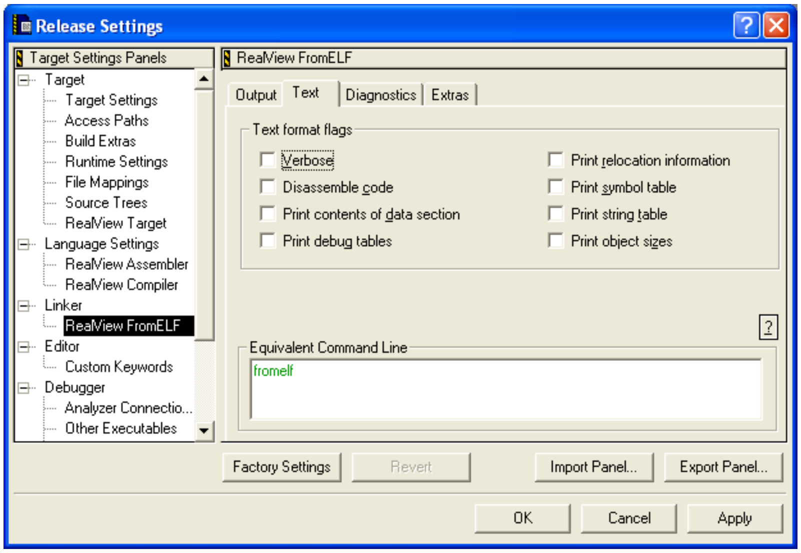


Figure 10-36 RealView fromELF Text panel

3. Select one or more text format flags to choose the information you want to include in the text file.

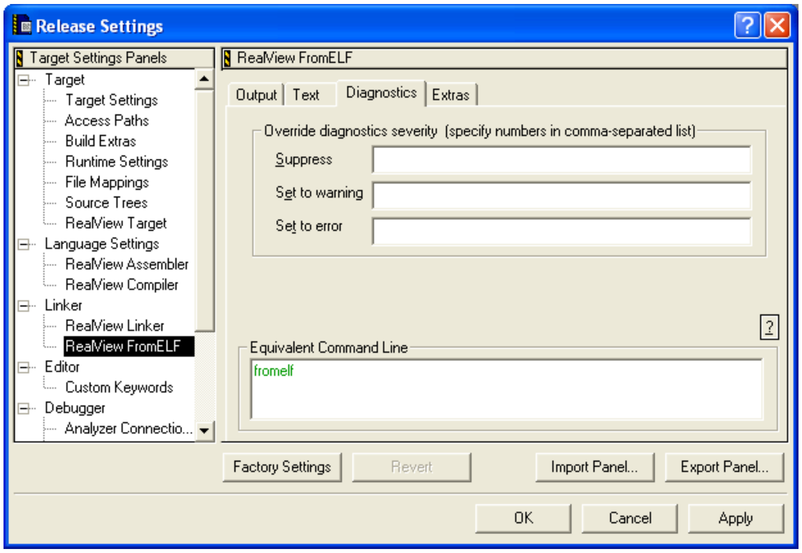
4. Click **Apply** to apply your changes.

Configuring RealView fromelf diagnostics

To configure the RealView `fromelf` diagnostics:

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 10-76).



Override diagnostics severity (specify numbers in comma-separated list)

Suppress

Set to warning

Set to error

Equivalent Command Line

```
fromelf
```

Figure 10-37 RealView fromELF Diagnostics panel

3. In the **Override diagnostics severity** group box, select values for the following options, as required:

   Suppress

   Disable diagnostic messages being produced for warnings with the specified tags. Enter a list of tags separated by commas.

   Set to warning

   Sets the diagnostic messages that have the specified tags to the warning severity.

Set to error

Sets the diagnostic messages that have the specified tags to the error severity.

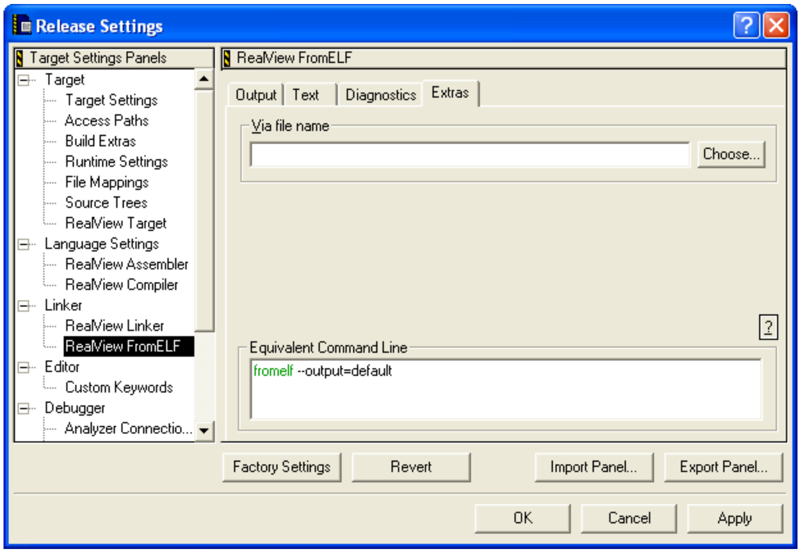
4. Click **Apply** to apply your changes.

Reading RealView fromelf options from a file

A *via* file is a text file that contains additional command-line arguments to the linker. Via files are read when the linker 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 linker settings are used by different build targets and projects. See the *RealView Compilation Tools Linker and Utilities 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. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 10-28 on page 10-63).



```
fromelf --output=default
```

Figure 10-38 RealView fromELF Extras 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.

10.7 Configuring editor settings

This section describes changes you can make to the CodeWarrior editor that apply to the current build target. See *Choosing editor preferences* on page 9-20 for information on setting global editor preferences.

10.7.1 Custom Keywords

The Custom Keywords settings panel enables you to set colors for defined sets of keywords. The colors you define are used to display the keywords in the CodeWarrior editor. Custom keywords are project-specific, not global to the CodeWarrior IDE. See *Text Colors* on page 9-31 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 10-39)

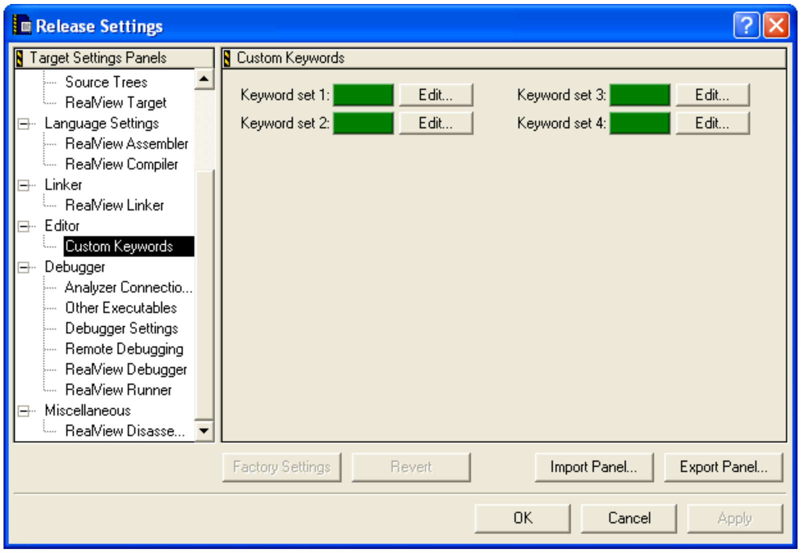


Figure 10-39 Custom Keywords settings panel

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.
4. Click **Apply** to apply your changes.

10.8 Configuring the debugger

You can configure the CodeWarrior IDE to call any of the following ARM debuggers to load and debug your images.

- RVD (RealView Debugger), the default debugger for new projects
- AXD (ARM eXtended Debugger)
- `armsd` (ARM Symbolic Debugger)

This section describes how to configure the ARM debuggers to debug and run executable images built from CodeWarrior IDE projects. See Chapter 5 *Working with the ARM Debuggers* for more information on how the CodeWarrior IDE interacts with the ARM debuggers.

For more information on using the debuggers described in this section, refer to the following guides:

- *RealView Debugger Essentials Guide* for detailed information on working with the ARM RealView debugger.
- *RealView Developer Suite AXD and armsd Debuggers Guide* for detailed information on working with the ARM eXtended Debugger (AXD) and the ARM symbolic debugger (`armsd`).

For information on configuring the other tools in the ARM tool chain see:

- *Configuring assembler and compiler language settings* on page 10-35
- *Configuring linker settings* on page 10-64.

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.

This section describes:

- *Remote Debugging* on page 10-84
- *Configuring the ARM Debuggers* on page 10-84
- *Configuring the RealView Runner* on page 10-90.

10.8.1 Analyzer Connections

The Analyzer Connections panel is not supported in CodeWarrior for RVDS.

10.8.2 Other Executables

The Other Executables panel is not supported in CodeWarrior for RVDS.

10.8.3 Debugger Settings

The Debugger Settings panel is not supported in CodeWarrior for RVDS.

10.8.4 Remote Debugging

The Remote Debugging panel is not supported in CodeWarrior for RVDS.

10.8.5 Configuring the ARM Debuggers

Use the Choose Debugger panel to configure options for the ARM debugger that the CodeWarrior IDE calls when you select **Debug** from the **Project** menu. You can call any of the ARM debuggers shown in the panel. Depending on the debugger, you can provide startup and configuration options, or specify a script file to call when the debugger is executed.

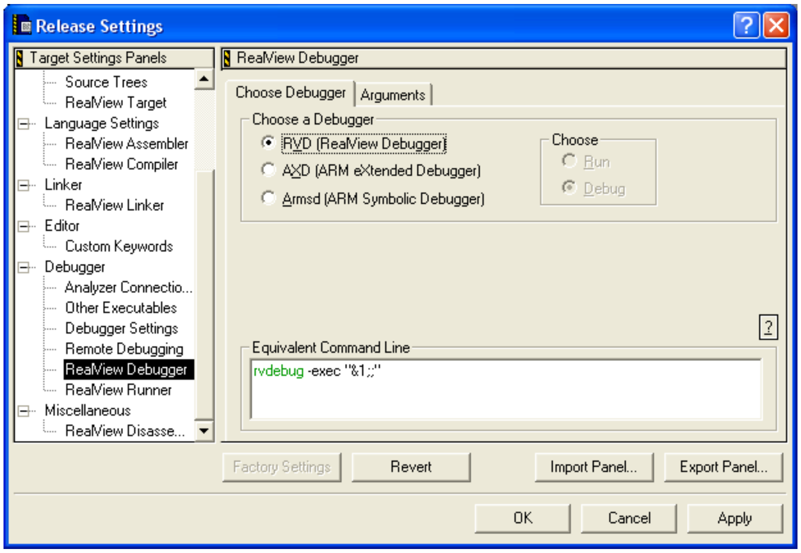
The following sections describe:

- *Choosing a debugger*
- *Configuring armsd* on page 10-86
- *Specifying arguments for your executable image* on page 10-89.

Choosing a debugger

Use the Choose Debugger panel to specify which ARM debugger is called to debug or run output images from the CodeWarrior IDE. To select the debugger called by the CodeWarrior IDE:

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 Debugger** in the Target Settings Panels list and click the **Choose Debugger** tab to display the configuration panel (Figure 10-40 on page 10-85).



Choose a Debugger: [✓ RVD (RealView Debugger)] AXD (ARM eXtended Debugger) Armsd (ARM Symbolic Debugger)

Choose: Run [✓ Debug]

Equivalent Command Line:

```
rvdebug -exec "&1;;"
```

Figure 10-40 Choose debugger panel

3. Choose the debugger you want to use from the **Choose Debugger** group box:
   - RVD (RealView Debugger)
   - AXD (ARM Extended Debugger); you can choose to start this debugger in Debug or Run mode from the **Choose** group box.
   - `armsd` (ARM Symbolic Debugger); you can choose to start this debugger in Debug or Run mode from the **Choose** group box.

   This is the debugger that will be called from the CodeWarrior IDE when you select **Debug** from the **Project** menu. Click one of the radio buttons and configure the debugger in the appropriate panel.

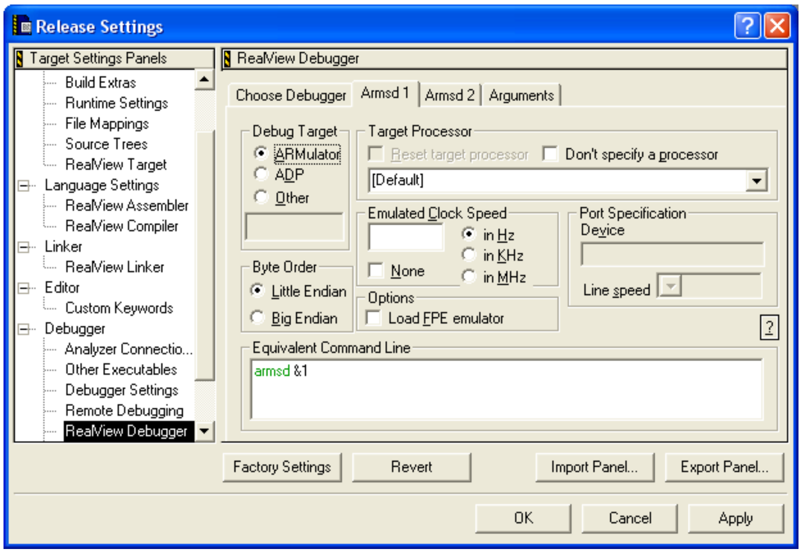
4. Configure the debugger you have chosen:
   - If your image requires them, you can specify arguments for all the debuggers in the **Arguments** tab. See *Specifying arguments for your executable image* on page 10-89.
   - Configure `armsd` using the **Armsd 1** and **Armsd 2** tabs. For more information on configuring `armsd` see *Configuring armsd* on page 10-86.

5. Click **Apply** to apply your settings.

Configuring armsd

If you have configured the CodeWarrior IDE to use `armsd` as your debugger (see *Choosing a debugger* on page 10-84), use the two `armsd` panels to configure debugger 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 Debugger** in the Target Settings Panels list and click the **Armsd1** tab to display the first configuration panel (Figure 10-41).



RealView Debugger

Debug Target: [✓ ARMulator] ADP Other

Target Processor: ☐ Reset target processor ☐ Don't specify a processor [Default]

Emulated Clock Speed: ☐ None [✓ in Hz] in KHz in MHz

Port Specification Device; Line speed

Byte Order: [✓ Little Endian] Big Endian

Options: ☐ Load FPE emulator

Equivalent Command Line: armsd &1

Figure 10-41 Armsd1 configuration panel

3. Select the debug target for the debugger:

ARMulator

Select **Armulator**® to target an ARMulator processor.

ADP	Select **ADP** (Angel Debug Protocol) if you are connecting to an ADP-compatible remote target such as an ARM development board. If you select this option you can specify the port and line speed for your target device.
Other	Select **Other** if you are connecting to a third-party debug target.

——— **Note** ———

You cannot use `armsd` with Multi-ICE® or RealView ICE.

———————————

4. Select values for the following options, as required:

Target Processor

Select the processor for your target system from the pop-up list. For ADP targets, select the **Reset target processor** checkbox to instruct the debugger to reset the target processor, if this is supported by the target system. Select **Don't specify a processor** to accept the default.

Byte Order

Set the byte order used by your target system. The image you are debugging must be compiled and assembled with the same byte order settings.

Load FPE emulator

Use this option if you want to run code compiled for a processor with a *Floating-Point Accelerator* (FPA) on a processor with no FPA, by using a *Floating-Point Emulator* (FPE). For example, use this option if you want to run code built on the ARM7500FE on the StrongARM.

Emulated Clock Speed

Select a clock speed for the ARMulator. ARMulator uses this value to convert cycle counts to time. If there is an `armsd.map` file, ARMulator uses information from this file in combination with the emulated clock speed to calculate both cycle count and time. The `armsd.map` file must be located in the same directory as `armsd`. See the *RealView Developer Suite AXD and armsd Debuggers Guide* for more information on map files.

Port Specification

If you have selected an ADP debug target, enter an expression in the Device text field to select the target communication method. The expression selects serial, serial/parallel, or ethernet communications and can be one of:

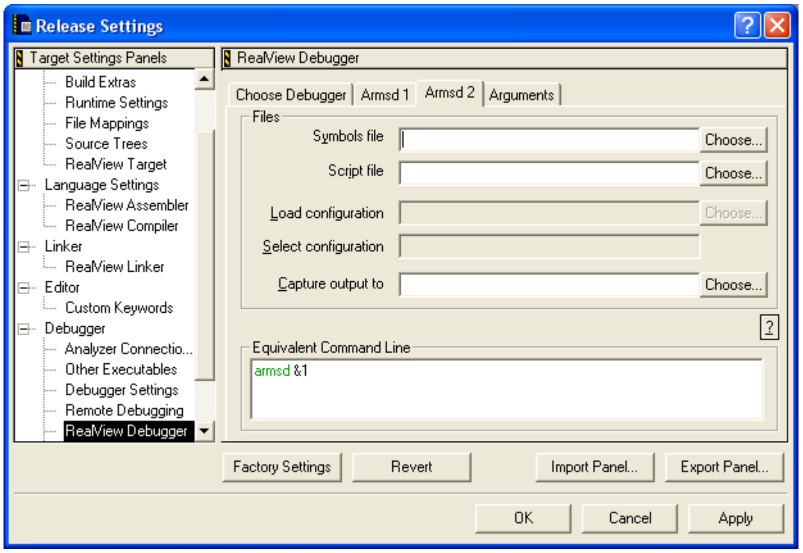
`s=`*`n`*	selects serial port communications. *n* can be 1, 2 or a device name.
`s=`*`n`*`,p=`*`m`*	selects serial and parallel port communication. *n* and *m* can be 1, 2, or a device name. There must be no space between the arguments.
`e=`*`id`*	selects ethernet communication. *`id`* is the ethernet address of the target board.

For serial and serial/parallel communications, you can append `, h=0` to the port expression to switch off the heartbeat feature of ADP. For example:

```
s=1,h=0
```

selects serial port 1 and turns off the ADP heartbeat.

5. Click the **Armsd2** tab to display the second configuration panel (Figure 10-42).



RealView Debugger

Symbols file
Script file
Load configuration
Select configuration
Capture output to

Equivalent Command Line

armsd &1

Figure 10-42 Armsd2 configuration panel

6. Enter values for the following options:

Symbols file

Enter the full pathname to an image file. `armsd` reads debug information from the image file, but does not load the image. Alternatively, click **Choose…** to select a symbols file from the standard file dialog box.

Script file

Enter the full path to a script file containing `armsd` commands that you want to execute on startup. Alternatively, click **Choose…** to select a script file from the standard file dialog box.

Load configuration

Enter the full path to an EmbeddedICE™ configuration file. Alternatively, click **Choose…** to select a configuration file from the standard file dialog box. Use the Select configuration field to select a specific configuration block from this file. This option is grayed out unless you have enabled ADP by selecting **ADP** under **Debug Target** in the **Armsd1** tab.

Select configuration

Enter an `armsd selectconfig` command to select a data block from the configuration file specified in the Load configuration field. An EmbeddedICE configuration data file contains data blocks, each identified by a processor name and version. This option is grayed out unless you have enabled ADP by selecting **ADP** under **Debug Target** in the **Armsd1** tab.

The `selectconfig` command selects the required block of EmbeddedICE configuration data from those available in the specified configuration file. See the `armsd` chapter of the *RealView Developer Suite AXD and armsd Debuggers Guide* for more information.

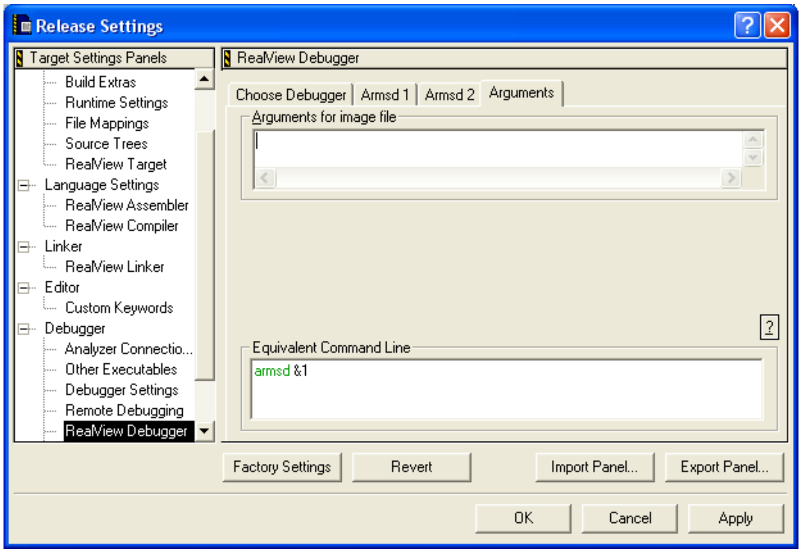
Capture output to

Enter a filename to which output information from the debugger is written. Alternatively, click **Choose…** to select an output file from the standard file dialog box.

7. Click **Apply** to apply your settings.

Specifying arguments for your executable image

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 Debugger** in the Target Settings Panels list.
3. Click the **Arguments** tab to display the arguments configuration panel (Figure 10-43 on page 10-90). Use this panel to enter any command-line arguments required by your executable image.



RealView Debugger

Equivalent Command Line: armsd &1

Figure 10-43 Arguments panel

4. Click **Apply** to apply your settings.

10.8.6 Configuring the RealView Runner

The term *RealView Runner* refers to the ARM debugger that is called to execute, rather than debug, an image file.

The RealView Runner panel is used to configure the debugger that is called when you select **Run** from the **Project** menu in the CodeWarrior IDE. You can use any of the ARM Debuggers to run executable images. You can specify different debuggers to be called when you debug, and when you run. For example, you can use AXD to debug your image, and `armsd` to run it without the overhead of starting a GUI debugger.

The options for this panel are exactly the same as for the RealView Debugger panel. See *Configuring the ARM Debuggers* on page 10-84 for information.

10.9 Configuring Miscellaneous features

This section describes the following miscellaneous configuration options:

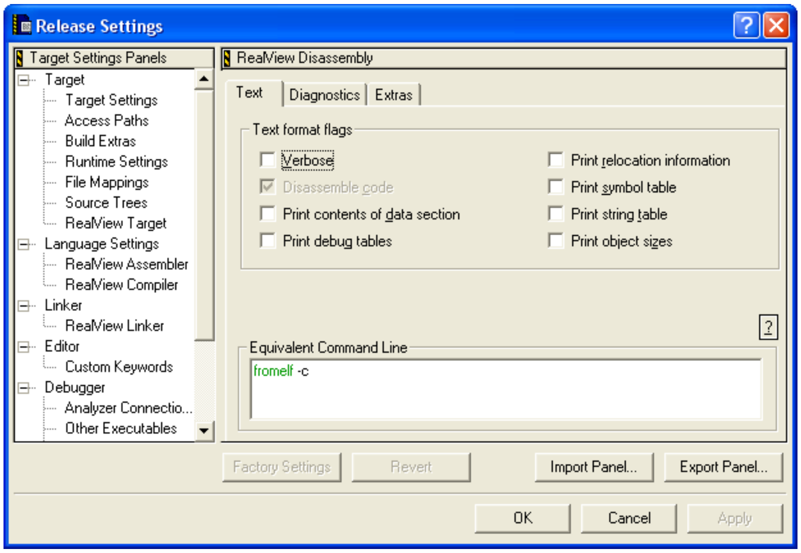
- *RealView Disassembly Features*.

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 or library file in the project window and choose **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 10-44).



Text format flags:
- [ ] Verbose
- [✓] Disassemble code
- [ ] Print contents of data section
- [ ] Print debug tables
- [ ] Print relocation information
- [ ] Print symbol table
- [ ] Print string table
- [ ] Print object sizes

Equivalent Command Line

```
fromelf -c
```

Figure 10-44 RealView Disassembly panel

3. Select the options you require from the Text format flags group box. The Disassemble code option is selected by default. For more information about these options refer to the --text option in the *RealView Compilation Tools Linker and Utilities Guide*.

4. Click on 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** panel. See *Configuring RealView fromelf diagnostics* on page 10-78.

5. Click on 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** panel. See *Reading RealView fromelf options from a file* on page 10-79.

6. Click **Apply** to apply your changes.

For more information about disassembling code, refer to *Disassembling code* on page 4-47.

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.

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.

If the CodeWarrior IDE has been installed in the default installation directory of `c:\Program Files\ARM`, then `CMDIDE.EXE` will be located in:

```
IDEs\CodeWarrior\CodeWarrior\5.6.1\1592\win_32-pentium\bin
```

(where `1592` is the build number). 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 proper parameters to produce a build, and waits for the 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 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 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 the project to build.

For example, to start CodeWarrior and load the `dhrystone` project, enter the following command:

```
“c:\Program
Files\ARM\IDEs\CodeWarrior\CodeWarrior\5.6.1\1592\win_32-pentium\bin\cmdide”
dhryansi.mcp /t Debug /c /b
```

(where `1592` is the build number). 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

Though `IDE.EXE` understands the same parameters as `CMDIDE.EXE`, 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.

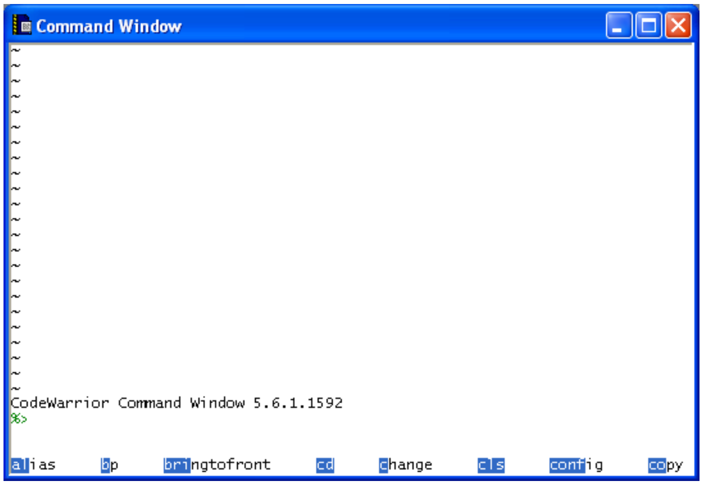
A.2 Using the CodeWarrior Command window

A command window can be opened from CodeWarrior and 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 and enter a command:

1. Choose **Command Window** from the **View** menu. The Command window is displayed (Figure A-1):



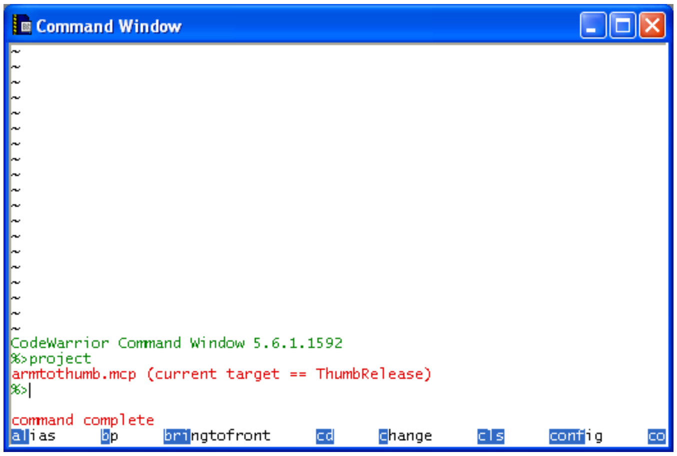
```
CodeWarrior Command Window 5.6.1.1592
%>

alias    bp    bringtofront    cd    change    cls    config    copy
```

Figure A-1 Command window

Commands can be entered at the command prompt, %>.

The list of available commands are shown at the bottom of the command window. Pressing the spacebar repeatedly shows all the commands which are available. The blue-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 (Figure A-2 on page A-5:



```
CodeWarrior Command Window 5.6.1.1592
%>project
armtothumb.mcp (current target == ThumbRelease)
%>|

command complete
alias     bp     bringtofront     cd     change     cls     config     co
```

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`.

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

The following table describes all the commands which are available from the CodeWarrior command window and indicates whether the command is supported for the CodeWarrior IDE for RVDS.

Table A-1 CodeWarrior commands from the Command window

Command	Description
alias	Create, remove or list alias.
bp	This command is not used by the CodeWarrior IDE for RVDS.
bringtofront	Console window always on top.
cd	Change directory.
change	This command is not used by the CodeWarrior IDE for RVDS.
cls	Clear screen.

Table A-1 CodeWarrior commands from the Command window (continued)

Command	Description
config	Configure the command window.
copy	This command is not used by the CodeWarrior IDE for RVDS.
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	This command is not used by the CodeWarrior IDE for RVDS.
display	This command is not used by the CodeWarrior IDE for RVDS.
evaluate	This command is not used by the CodeWarrior IDE for RVDS.
exit	Close the command window.
getpid	This command is not used by the CodeWarrior IDE for RVDS.
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	This command is not used by the CodeWarrior IDE for RVDS.
project	Open or close a project file.
pwd	Display current working directory.
quitIDE	Quit the CodeWarrior IDE.
radix	This command is not used by the CodeWarrior IDE for RVDS.
removeobj	Remove all binaries for the specified project or the default project if no project is specified.
reset	This command is not used by the CodeWarrior IDE for RVDS.
restart	Restart a debugging session.
restore	This command is not used by the CodeWarrior IDE for RVDS.
rget	This command is not used by the CodeWarrior IDE for RVDS.

Table A-1 CodeWarrior commands from the Command window (continued)

Command	Description
rset	This command is not used by the CodeWarrior IDE for RVDS.
save	This command is not used by the CodeWarrior IDE for RVDS.
sourcedisplay	This command is not used by the CodeWarrior IDE for RVDS.
stack	This command is not used by the CodeWarrior IDE for RVDS.
status	This command is not used by the CodeWarrior IDE for RVDS.
step	This command is not used by the CodeWarrior IDE for RVDS.
stop	This command is not used by the CodeWarrior IDE for RVDS.
switchtarget	This command is not used by the CodeWarrior IDE for RVDS.
system	Execute a system command.
view	This command is not used by the CodeWarrior IDE for RVDS.
wait	This command is not used by the CodeWarrior IDE for RVDS.
watchpoint	This command is not used by the CodeWarrior IDE for RVDS.
window	This command is not used by the CodeWarrior IDE for RVDS.

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-4
- *Using the cwfileassoc tool* on page B-5.

B.1 The CodeWarrior IDE preferences directory

The CodeWarrior IDE maintains preferences and persistence information in the following location in the registry:

```
c:\Documents and Settings\<username>\Local Settings\Application
Data\Metrowerks\Code Warrior IDE 5.5 Prefs
```

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 deleted when you uninstall the CodeWarrior IDE for RVDS.

B.2 The RVCT22BIN environment variable

CodeWarrior for RVDS uses the `RVCT22BIN` environment variable to locate the RVCT build tools, not the `PATH` environment variable. This is set up for you by the RVDS installer.

B.3 Using different versions of the CodeWarrior IDE

The CodeWarrior IDE for RVDS is customized to support the ARM tool chain. 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 the CodeWarrior `bin` subdirectory.

To switch back to the CodeWarrior IDE for RVDS, rename your preferences directories and run `regservers.bat` from the following location (assuming the default installation directory of `c:\Program Files\ARM`:

```
\IDEs\CodeWarrior\CodeWarrior\5.6.1\1592\win_32-pentium\bin.
```

(where `1592` is the build number).

B.4 Using the cwfileassoc tool

This `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.

This section describes the syntax of the `cwfileassoc` tool, and shows an example of how to use the tool. It includes:

- *cwfileassoc command syntax*
- *Example*

You can find the `cwfileassoc` tool at the following location (assuming the default installation directory of `c:\Program Files\ARM`:

```
install_directory\IDEs\CodeWarrior\CodeWarrior\5.6.1\1592
```

(where `1592` is the build number).

B.4.1 cwfileassoc command syntax

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.4.2 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.

Appendix C
CodeWarrior Project Conversion Utilities

This appendix describes the utilities which are available for converting projects or files created using other development environments into CodeWarrior for RVDS projects or exporting CodeWarrior RVDS projects into other file formats.

C.1 Introduction

CodeWarrior for RVDS provides utilities which can be used to convert any of the following projects or files into CodeWarrior for RVDS projects:

- CodeWarrior for ARM Developer Suite (ADS) projects; see *Converting CodeWarrior for ADS projects* on page C-3.
- GNU Makefiles; CodeWarrior can import GNU Makefiles as CodeWarrior projects and also export CodeWarrior projects as GNU Makefiles. See *Converting GNU Makefiles* on page C-6.
- RealView Debugger (RVD) project files via XML; see *Converting RVD project files* on page C-9.

The following sections describe how to perform the above conversions.

C.2 Converting CodeWarrior for ADS projects

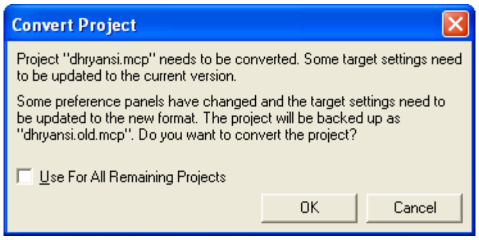
This section describes how to convert projects created using the CodeWarrior IDE for the ARM Developer Suite (ADS) projects.

If you have CodeWarrior for ADS projects, you can upgrade them to CodeWarrior for RVDS projects. To do this:

1. Select **File** → **Open...** from the main menu to display the Open dialog box.
2. Locate your CodeWarrior for ADS 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 new CodeWarrior for RVDS format.


Convert Project

Project "dhryansi.mcp" needs to be converted. Some target settings need to be updated to the current version.

Some preference panels have changed and the target settings need to be updated to the new format. The project will be backed up as "dhryansi.old.mcp". Do you want to convert the project?

Figure C-1 Convert Project dialog box

3. If you have selected multiple project files to convert, select **Use For All Remaining Projects** to update the target settings panels for all selected CodeWarrior for ADS project files. This also prevents the Convert Project dialog box being displayed again when CodeWarrior converts the remaining project files.
4. Click **OK** to convert the project. CodeWarrior for RVDS:
   - 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

CodeWarrior for ADS has four compiler panels, because in ADS the ARM and Thumb, C and C++ compilers were separate compilers. However, in RVDS the compilers are combined into a single executable. Therefore, in CodeWarrior for RVDS there is only one compiler panel.

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 used, the settings for all non-empty ADS panes 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 is displayed again for the next project.

Note

ADS 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 RVDS provides only the `Release` and `Debug` build targets (see *Predefined build targets* on page 4-4).

---

5. Some build tool options cannot be imported from the ADS project. If your ADS project uses options that cannot be imported, they are highlighted in red in the equivalent command line for the associated panel. To fix these, follow the instructions in *Fixing options that cannot be imported from ADS*.

C.2.1 Fixing options that cannot be imported from ADS

If your ADS 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 **Edit** → ***TargetName*** **Settings…** from the main menu.
2. Click **RealView Assembler** in the Target Settings Panels list. CodeWarrior for RVDS 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. CodeWarrior for RVDS displays the RealView Compiler panel.

5. Remove any options from the Equivalent Command Line that are highlighted in red.
6. Review the remaining compiler options in the equivalent command line and check that they are correct. In particular, if your ADS 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 ARM/Thumb State to **Thumb**.

      This adds the `--thumb` compiler option.
7. Click **RealView FromELF** in the Target Settings Panels list. CodeWarrior for RVDS 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 check that they are correct.
10. Click **OK** to save your changes and close the *TargetName* Settings dialog box.

    When you have made the appropriate changes to all the build targets, you can build your project.

C.3 Converting GNU Makefiles

This section describes how to convert GNU Makefiles into CodeWarrior projects using the Makefile Importer Wizard and also how to export CodeWarrior projects to GNU Makefiles.

C.3.1 Importing GNU makefiles into projects

The CodeWarrior IDE can import some simple Visual C `nmake` or GNU makefiles into CodeWarrior IDE project files. The CodeWarrior IDE uses the Makefile Importer wizard to process the files. The wizard performs the following tasks:

- parses the makefile
- creates a CodeWarrior IDE project
- creates build targets
- adds source files as specified in the makefile
- matches the information specified in the makefile to the output name, output directory, and access paths of the created build targets
- selects a linker to use with the project.

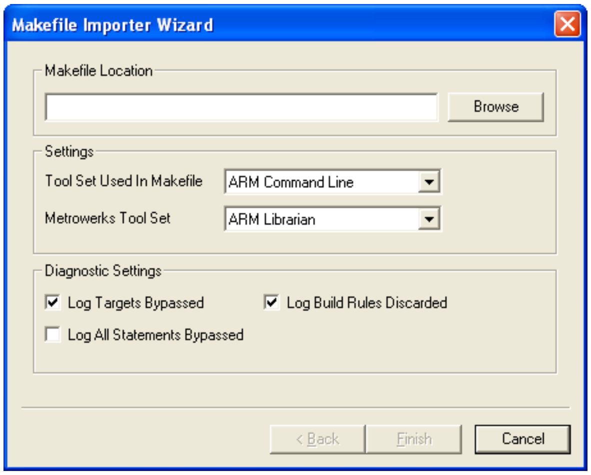
Note

The CodeWarrior Makefile Importer Wizard has some limits on the complexity of makefile that it can import. It might fail to convert complex makefiles, or makefiles that contain certain non-standard constructs. In addition, the Makefile Importer is *not* able to import makefiles that have been created by the Makefile Exporter. The use of the Makefile Importer is deprecated.

Using the Makefile Importer wizard

To create a new project from a makefile:

1. Select **New…** from the **File** menu. The CodeWarrior IDE displays the New dialog box.
2. Click the **Project** tab and select the **Makefile Importer Wizard** from the list of project stationery.
3. Enter a name and location for the project to be created from the imported makefile (see *Creating a new project* on page 3-13 for naming conventions) and click **OK**.The CodeWarrior IDE displays the Makefile Importer Wizard (Figure C-2 on page C-7).


Makefile Importer Wizard

Makefile Location

Settings

Tool Set Used In Makefile: ARM Command Line

Metrowerks Tool Set: ARM Librarian

Diagnostic Settings

[✓] Log Targets Bypassed

[✓] Log Build Rules Discarded

[ ] Log All Statements Bypassed

Figure C-2 Makefile Importer Wizard

4. Enter the location of the makefile in the Makefile Location text field, or click **Browse** to select the makefile from the standard file dialog.

5. Select Settings options:

   Tool Set Used in Makefile

   Select the makefile tool on which the makefile build rules are based from the drop-down list.

   Metrowerks Tool Set

   Select **ARM Linker** from the drop-down list.

6. Select Diagnostic Settings as required:

   Log Targets Bypassed

   Select this option to log information about the build targets parsed in the makefile that were not converted to CodeWarrior IDE build targets.

   Log Build Rules Discarded

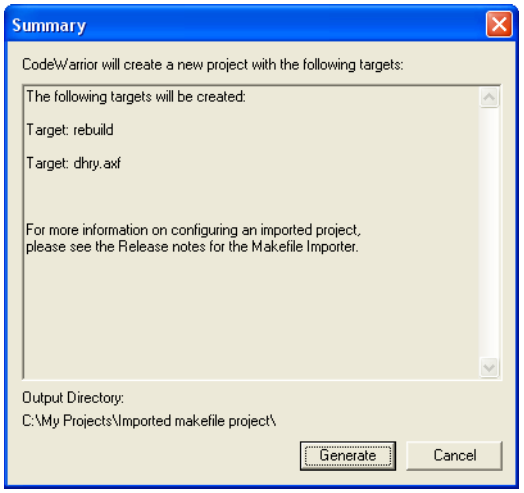
   Select this option to log information about the build rules in the makefile that were discarded in the conversion to a CodeWarrior IDE project.

   Log All Statements Bypassed

   Select this option to log the same information as the Log Targets Bypassed and Log Build Rules Discarded options, and information about other items in the makefile that were not understood during the parsing process.

Diagnostic messages are displayed in a project message window. The project message window is similar to the message window. See *Using the message window* on page 5-14 for more information.

7. Click **Finish**. If the Makefile Importer wizard can successfully process the makefile, it displays a summary window showing the current conversion settings (Figure C-3).



CodeWarrior will create a new project with the following targets:

The following targets will be created:

Target: rebuild

Target: dhry.axf

For more information on configuring an imported project, please see the Release notes for the Makefile Importer.

Output Directory:
C:\My Projects\Imported makefile project\

Figure C-3 Makefile importer summary

8. Click **Generate** to import the makefile. The CodeWarrior IDE generates a new project based on the makefile.

C.3.2 Exporting projects to GNU Makefiles

This section describes how to export CodeWarrior project files to GNU makefiles. To do this:

1. Open the CodeWarrior project you wish to convert.

2. Choose **Export Project as GNU Makefile** from the **Project** menu.

   The project is converted. Once the conversion is complete a confirmation message is displayed in the Project Messages window. The format of the message is:

   ```
   <projectname>.mcp exported successfully to <pathname>/<project name>.mk
   ```

   The makefile is saved in the same location as the CodeWarrior project file.

C.4 Converting RVD project files

All types of RealView Debugger (RVD) projects can be converted to CodeWarrior projects. The conversion procedure to follow depends on the type of RVD project to be converted:

- Standard and Library RVD projects; see *Converting RVD Standard and Library projects*
- RVD Container projects; see *Converting RealView Debugger Container projects* on page C-12.
- RVD Custom projects; see *Converting RealView Debugger Custom projects* on page C-12.

C.4.1 Converting RVD Standard and Library projects

To convert RVD 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 CodeWarrior.

This section includes:

- *Converting a RealView Debugger project file to XML format*
- *Example conversion of dhrystone.prj* on page C-11
- *Importing XML formatted project files* on page C-12

For more information on importing and exporting XML files in CodeWarrior, refer to *Importing and exporting a project as XML* on page 3-20.

Converting a RealView Debugger project file to XML format

To convert a RealView Debugger project file to XML, run the `prj2xml` command and specify the RealView Debugger project file to be converted.

The `prj2xml` command enables you to convert one or more RealView Debugger project files to CodeWarrior for RVDS XML formatted files. The syntax of the 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.

If the RVD project includes relative pathnames and the CodeWarrior project is created in a different location, these paths will be broken. If you wish to create the CodeWarrior project in a separate directory from the original RVD project then you need to either:

- Replace any relative paths in the RVD project with absolute paths prior to saving the project as XML
- Replicate the directory strucutre in the new location.

The CodeWarrior 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 build target names are:

- `arm_cpp_Debug`
- `arm_cpp_Release`.

Table C-1 shows the options supported by the 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-11 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. By default, the `template.xml` file supplied with `prj2xml` is used.
`--spec=`*`filename`*	The RealView Debugger project file specification you want to use. By default, the `spec.stp` file supplied with `prj2xml` is used.

Note

The two options shown in Table C-1 on page C-10 (`--template` and `--spec`) enable you to override the default configuration files, and are not usually required. If you have to use these options, you must use them with care.

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.

Importing XML formatted project files

To import an XML formatted project file:

1. Select **File** → **Import Project...** from the main 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 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 RVD Container projects. (RVD Container are projects which contain other RVD subprojects.) If you wish to convert an RVD Container project then you must treat each sub-project in the RVD Container project as a separate project. The procedure to follow is:

1. Locate each subproject (`.prj` file) of the RVD Container project.
2. Import each `.prj` file of the RVD Container project into CodeWarrior using the `prj2xml` utility (as described in the previous two sections).
3. Make a new CodeWarrior project and add all the converted subprojects to it.

   For more information on creating subprojects in CodeWarrior, refer to *Working with multiple build targets and subprojects* on page 3-44.

C.4.3 Converting RealView Debugger Custom projects

To convert RealView Custom projects you must import the project makefile directly using the CodeWarrior Makefile Importer Wizard. For more information, refer to *Converting GNU Makefiles* on page C-6.

Appendix D
CodeWarrior IDE Reference

This chapter describes each menu command in the CodeWarrior IDE, and the default key bindings for those commands. You can use this chapter 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.

D.1 CodeWarrior IDE menu reference

This section gives an overview of the menu commands in the CodeWarrior IDE. The following sections describe the menus in the CodeWarrior IDE 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
- *Toolbar submenu* on page D-15.
- *System submenu* on page D-16

The lists below summarize the menus that are displayed at all times, and which menus are displayed only when a window that can use their menu commands is available.

The following menus are always available:

- **File**
- **Edit**
- **View**
- **Search**
- **Project**
- **Debug** (this menu is not used by the CodeWarrior IDE for RVDS)
- **Tools** (this menu is not used by the CodeWarrior IDE for RVDS)
- **Window**
- **Help**.

The following menus are context-sensitive:

- **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 System (VCS)** menu is displayed if you have installed and configured the CodeWarrior IDE to work with a compatible revision control system that you purchased separately. Refer to the Metrowerks web site for more information about using a VCS with CodeWarrior.

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 Menu options from the File menu

Menu option	Description
New	This command opens the **New** dialog box. This dialog box 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 3-25 for more information.
Close Workspace	This command closes the current workspace. See *Closing workspaces* on page 3-26 for more information.
Save Workspace	This commands saves the current workspace. See *Saving the current workspace or creating a new workspace* on page 3-25 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 3-25 for more information.

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 3-20 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 3-20 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.

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 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 9-23 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 *Other settings* on page 9-11 in the *Editor settings* on page 9-23 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 messages window is active, the **Copy** command copies all the text in the messages 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 messages window, the **Paste** menu item is grayed out and cannot be executed.
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.

Table D-2 Menu options from the Edit menu (continued)

Menu option	Description
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 on page 6-15 on page 6-15*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.
[Target] 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 Settings window. See the menu option, **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 **Version Control System** (VCS) options panel. (For more information about using the CodeWarrior IDE with a VCS refer to the Metrowerks web site.)
Commands and Key Bindings...	Use this menu item to set keybinding for commands, and to customize the CodeWarrior IDE toolbars.

D.1.3 View menu

The **View** menu contains commands for viewing toolbars, browsers, and windows.

Table D-3 Menu options from the View menu

Menu option	Description
Toolbars	This command causes the **Toolbar** submenu to appear. See *Toolbar 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 allows 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 view* 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 view* 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 4-41.
Errors and Warnings	This command displays the Errors and Warnings window. See *Using the message window* on page 5-11 for more information. See also the Find All section in *Finding and replacing text with the Find and Replace dialog* 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. See
Breakpoints	This menu item is not used by the CodeWarrior IDE for RVDS.

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 ARM Debuggers* for more information.
Expressions	This window is not used by the CodeWarrior IDE for RVDS. See Chapter 5 *Working with the ARM Debuggers* for more information.
Global Variables	This window is not used by the CodeWarrior IDE for RVDS. See Chapter 5 *Working with the ARM Debuggers* 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 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. This command is useful if you want to replace one instance of a text string without having to open the Find window. For example, if you have just replaced all the occurrences of the variable `icount` with `jcount` and discover an instance of `icont`, you can replace this variable with `jcount` by selecting **Replace** from the **Search** menu. See *Finding and replacing text with the Find and Replace dialog* 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. For more information, see Chapter 7 *Searching and Replacing Text*.
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* on page 7-4 for more information.

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* 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. For example, if you have just replaced all the occurrences of the variable `icount` with `jcount` and discover an instance of `icont`, you can replace this variable with `jcount` by selecting **Replace Selection** from the **Search** menu. 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* 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 messages window appears warning you of multiple definitions. For more information on the messages window, see *Using the message window* on page 5-14. 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.

Table D-4 Menu options from the Search menu (continued)

Menu option	Description
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.
Go to Line...	This command opens a 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 dialog box to choose two files or folders to compare and merge. After choosing files to compare, a file comparison window appears, showing differences between the two files. If two folders are compared, the differences between the folders are shown in the Compare Folders window. See *Comparing and merging files and folders* 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 allows you to add and remove files and libraries from your project. It also allows you to compile, build, and link your project. All of these commands are described in this section.

Table D-5 Menu options from the Project menu

Menu option	Description
Add [Window]	This command adds the file in the active editor window to the open project. See *Adding the current editor window* on page 3-32 for more information.
Add Files...	This command adds files to the project window. See *Using the Add Files command* on page 3-30 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 3-33 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 subprojects* on page 3-44 for more information.

Table D-5 Menu options from the Project menu (continued)

Menu option	Description
Create Segment/Overlay	Creates a new segment or overlay in the current project. The IDE enables this menu command after you click the Segments tab or the Overlays tab in the active Project window. See Chapter 3 *Working with Projects* for more information.
Create Design	This menu item is not used by the CodeWarrior IDE for RVDS
Export Project as GNU Makefile	This command exports CodeWarrior projects to GNU makefiles. See on page C-8*Exporting projects to GNU Makefiles* on page C-8.
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 messages window appears. For information on how to correct compiler errors, consult *Correcting compilation errors and warnings* on page 5-16.
Preprocess	This command performs preprocessing on selected C and C++ source code files. See *Preprocessing source code* on page 4-43 for more information.
Precompile	This menu option 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 segments/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 4-38 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 dialog box. This is accessed using the **Commands and Key Bindings...** command in the **Edit** menu. See *Setting commands and key bindings* on page 9-38 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 4-47 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 4-42 for more information.

Table D-5 Menu options from the Project menu (continued)

Menu option	Description
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 4-44 for more information.
Stop Build	This command stops the current make operation.
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 4-45 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 3-38 for more information.
Debug	This command compiles and links the current build target, and launches an ARM debugger to debug the output image.
Run	This command compiles and links the current build target, and launches an ARM 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 3-19 for more information.
Set Default Target	This command enables you to choose a different default target.

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 present when a browser window is open. Otherwise, the menu is not displayed.

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... and 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 Menu options from the Window menu

Menu option	Description
Close	Closes the active window.
Close All	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 messages window.
Tile Horizontally	Arranges open editor windows horizontally so that none overlap.

Table D-7 Menu options from the Window menu (continued)

Menu option	Description
Tile Vertically	Arranges open editor windows vertically so that none overlap.
Save Default Window	Saves the settings of the active browser window. The CodeWarrior IDE applies the saved settings to subsequently opened browser windows.
[Other window menu items]	The other **Window** menu items depend solely 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 the Control key 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. When you are working in the CodeWarrior IDE, select one of the items to get interactive, online help.

——— **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. However, it does provide useful, general information about the features of CodeWarrior.

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.

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 Toolbar submenu

The **View** menu has a submenu under it for the **Toolbar** command. The **Toolbar** submenu contains all the commands used to customize the toolbars that appear in CodeWarrior IDE windows. See *Customizing toolbars* on page 9-48 for more information.

Table D-9 Menu options from the View > Toolbar 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 actual 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 Elements window. 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 actual 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 Elements window.

D.1.10 System submenu

The **View** menu has a submenu under it for the **System** command. The **System** submenu contains a single menu entry.

Table D-10 Menu options from the View > 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.

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 9-38. The commands in this section are listed in the order that they appear in the Windows layout.

The key bindings sections include:

- *File menu*
- *Edit menu* on page D-18
  View menu on page D-19
- *Search menu* on page D-19
- *Project menu* on page D-20
- *Window menu* on page D-21
- *Miscellaneous* on page D-21
- *Editor commands* on page D-22.

Note

Menu commands which are not shown in these sections do not have default key bindings assigned to them.

D.2.1 File menu

Table D-11 lists the default key bindings for manipulating projects and files in the CodeWarrior IDE.

Table D-11 File key bindings

Command	Key binding
New	Ctrl-N
New…	Ctrl-Shift-N
Open	Ctrl-O
Find and Open File	Ctrl-Shift-D
Close	Ctrl-W
Close All	Ctrl-Shift-W

Table D-11 File key bindings (continued)

Command	Key binding
Save	Ctrl-S
Save All	Ctrl-Shift-S
Print	Ctrl-P

D.2.2 Edit menu

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-.
[Target] Settings	Alt-F7

D.2.3 View menu

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-1

D.2.4 Search menu

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-H
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

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 menu

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 menu

Table D-16 contains the default key bindings for handling many common windows in the CodeWarrior IDE.

Table D-16 Window menu key bindings

Command	Key binding
Close	Ctrl-W
Close All	Ctrl-Shift-W
Select Default Project	Ctrl-0
Select Document 1	Ctrl-1
Select Document 2	Ctrl-2
Select Document 3	Ctrl-3
Select Document 4	Ctrl-4
Select Document 5	Ctrl-5
Select Document 6	Ctrl-6
Select Document 7	Ctrl-7
Select Document 8	Ctrl-8
Select Document 9	Ctrl-9

D.2.7 Miscellaneous

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
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
Delete to End of File	-
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

Table D-18 Editor window key bindings (continued)

Command	Key binding
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
Scroll Page Up	-
Scroll Page Down	-
Scroll to Top of File	-
Scroll to End of File	-
Scroll to Selection	-
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

Glossary

AAPCS	See *Procedure Call Standard for the ARM Architecture*.
ANSI	American National Standards Institute. An organization that specifies standards for, among other things, computer software.
ARM Developer Suite (ADS)	A suite of software development applications, together with supporting documentation and examples that enables you to write and debug applications for the ARM family of RISC processors. ADS is superceded by the RealView Developer Suite (RVDS). *See also* RealView Developer Suite.
ARM eXtended Debugger (AXD)	The ARM eXtended Debugger (AXD) is a single-processor debugger that runs on Windows platforms. AXD supports Multi-ICE and RealView ARMulator ISS debug targets for ARM7 and ARM9 processors only. *See also* ARM Symbolic Debugger, Multi-ICE, RealView ARMulator ISS, and RealView Debugger.
ARM Symbolic Debugger (armsd)	*ARM Symbolic Debugger* (`armsd`) is a command-line debugger that runs on all supported platforms. `armsd` supports only RealView ARMulator ISS debug targets with legacy ARM7 and ARM9 processors only. *See also* ARM eXtended Debugger, RealView ARMulator ISS, and RealView Debugger.
armar	The ARM RealView librarian.

armasm	The ARM RealView assembler.
armcc	The ARM RealView C/C++ compiler.
armlink	The ARM linker.
armsd	*See* ARM Symbolic Debugger.
ARMulator	ARMulator is an instruction set simulator. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors.
ATPCS	ARM and Thumb Procedure Call Standard (ATPCS) defines how registers and the stack will be used for subroutine calls. *See also Procedure Call Standard for the ARM Architecture*.
AXD	*See* ARM eXtended Debugger.
Big-endian	Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word. *See also* Little-endian.
Coprocessor	Additional hardware which is used for certain operations. Usually used for floating-point math calculations, signal processing, or memory management.
Debug With Arbitrary Record Format (DWARF)	ARM code generation tools generate debug information in DWARF2 format by default. From RVCT v2.2, you can optionally generate DWARF3 format (Draft Standard 9).
Debugger	An application that monitors and controls the execution of a second application. Usually used to find errors in the application program flow.
Deprecated	A deprecated option or feature is one that you are strongly discouraged from using. Deprecated options and features will not be supported in future versions of the product.
DWARF	*See* Debug With Arbitrary Record Format.
ELF	*See* Executable and linking format.
EmbeddedICE logic	The EmbeddedICE logic is an on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface. *See also* IEEE1149.1.
Environment	The actual hardware and operating system that an application will run on.

Executable and linking format (ELF)	The industry standard binary file format used by the RealView Developer Suite. ELF object format is produced by the ARM object producing tools such as `armcc` and `armasm`. The RealView linker accepts ELF object files and can output either an ELF executable file, or partially linked ELF object.
Execution view	The address of regions and sections after the image has been loaded into memory and started execution.
Host	A computer which provides data and other services to another computer.
IDE	Integrated Development Environment (the CodeWarrior IDE).
Image	An executable file which has been loaded onto a processor for execution. A binary execution file loaded onto a processor and given a thread of execution. An image can have multiple threads. An image is related to the processor on which its default thread runs.
Inline	Functions that are repeated in code each time they are used rather than having a common subroutine. Assembler code placed within a C or C++ program.
Input section	Contains code or initialized data or describes a fragment of memory that must be set to zero before the application starts.
Interworking	Producing an application that uses both ARM and Thumb code.
Joint Test Action Group (JTAG)	An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a *Joint Test Action Group* (JTAG) interface port to communicate with processors. For further information refer to IEEE Standard, Test Access Port and Boundary-Scan Architecture specification 1149.1 (JTAG).
JTAG	*See* Joint Test Action Group.
Library	A collection of assembler or compiler output objects grouped together into a single repository.
Linker	Software which produces a single image from one or more source assembler or compiler output objects.
Little-endian	Memory organization where the least significant byte of a word is at a lower address than the most significant byte. *See also* Little-endian.
Local	An object that is only accessible to the subroutine that created it.
Multi-ICE	A JTAG-based tool for debugging embedded systems.

Output section
Is a contiguous sequence of input sections that have the same RO, RW, or ZI attributes. The sections are grouped together in larger fragments called regions. The regions will be grouped together into the final executable image.

See also Region

Procedure Call Standard for the ARM Architecture
The Procedure Call Standard for the ARM Architecture specifies a family of *Procedure Call Standard* (PCS) variants to define how separately compiled and assembled routines can work together. The standard provides equal support for both ARM state and Thumb state to enable interworking. It favors small code size and provides functionality appropriate to embedded applications and high performance.

Read Only Position Independent
A section in which code and read-only data addresses can be changed at run-time.

Read Write Position Independent
A section in which read/write data addresses can be changed at run-time.

RealView ARMulator ISS (RVISS)
The most recent version of the ARM simulator, RealView ARMulator ISS is supplied with RealView Developer Suite. It communicates with a debug target using RV-msg, through the RealView Connection Broker interface, and RDI.

See also RDI and RealView Connection Broker.

RealView Compilation Tools (RVCT)
RealView Compilation Tools is a suite of tools, together with supporting documentation and examples, that enables you to write and build applications for the ARM family of *RISC* processors.

RealView Debugger
RealView Debugger is a multiprocessor debugger that runs on all supported platforms. You can connect to debug targets using Multi-ICE, RVISS, and RealView ICE.

See also ARM Symbolic Debugger, Multi-ICE, RealView ARMulator ISS, and RealView ICE.

Remote Debug Interface (RDI)
The *Remote Debug Interface* (RDI) is an ARM standard procedural interface between a debugger and the debug agent. RDI gives the debugger a uniform way to communicate with:

- a simulator running on the host (for example, RVISS)
- a debug monitor running on ARM architecture-based hardware accessed through a communication link (for example, Angel Debug Protocol)
- a debug agent controlling an ARM processor through hardware debug support (for example, Multi-ICE).

RealView Developer Suite
A suite of applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of RISC processors.

RealView ICE (RVI)	A JTAG-based debug solution to debug software running on ARM processors.
Regions	In an Image, a region is a contiguous sequence of one to three output sections (RO, RW, and ZI).
RISC	Reduced Instruction Set Computer.
RO	Read-only.
ROPI	See *Read Only Position Independent*.
RVCT	*See* RealView Compilation Tools.
RVDS	See *RealView Developer Suite*.
RVISS	*See* RealView ARMulator ISS.
RWPI	See *Read Write Position Independent*.
Scatter loading	Assigning the address and grouping of code and data sections individually rather than using single large blocks.
Scope	The accessibility of a function or variable at a particular point in the application code. Symbols which have global scope are always accessible. Symbols with local or private scope are only accessible to code in the same subroutine or object.
Section	A block of software code or data for an Image. *See also* Input sections
Semihosting	A mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself.
Target	The actual target processor, (real or simulated), on which the application is running. The fundamental object in any debugging session. The basis of the debugging system. The environment in which the target software will run. It is essentially a collection of real or simulated processors.
Thread	A context of execution on a processor. A thread is always related to a processor and might or might not be associated with an image.
VCS	Version Control System.
Word	A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.
ZI	Read/Write memory used to hold variables that do not have an initial value. The memory is normally set to zero on reset.