ARM® Developer Suite

Version 1.1

CodeWarrior® IDE Guide

ARM

ARM DUI 0065C

CodeWarrior IDE Guide
Version 1.1

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Change
October 1999	A	Release 1.0
March 2000	B	Release 1.0.1
November 2000	C	Release 1.1

Conformance Notice

Year 2000 Conformance

The Products provided by Metrowerks under the License agreement process dates only to the extent that the Products use date data provided by the host or target operating system for date representations used in internal processes, such as file modifications. Any Year 2000 Compliance issues resulting from the operation of the Products are therefore necessarily subject to the Year 2000 Compliance of the relevant host or target operating system. Metrowerks directs you to the relevant statements of Microsoft Corporation, Sun Microsystems, Inc., Apple Computer, Inc., and other host or target operating systems relating to the Year 2000 Compliance of their operating systems. Except as expressly described above, the Products, in themselves, do not process date data and therefore do not implicate Year 2000 Compliance issues.

For additional information, visit: http://www.metrowerks.com/about/y2k.html

Contents
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 xiii.

About this book

This book provides user information for *CodeWarrior for the ARM 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 development projects under Windows NT, 95, 98, or 2000. It assumes that you are an experienced software developer, and that you are familiar with the ARM development tools, as described in *Getting Started*. 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 ARM development tools.

Chapter 2 ***Working with Projects***

Read this chapter for details of how to use CodeWarrior 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 a CodeWarrior project.

It also describes the ARM project stationery provided with the CodeWarrior IDE, and describes how to use and modify the default stationery to generate ARM and Thumb® executable images, libraries, and disassembled code listings.

Chapter 3 ***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 4 ***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 5 ***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 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 6 ***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 7 ***Browsing Source Code***

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 8 ***Configuring the IDE***

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 9 ***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 ARM compilers, 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 postlinker to use, and the access paths and file mappings that apply to a build target.

Chapter 10 ***Using CodeWarrior IDE with Version Control Systems***

Read this chapter for general information on using the CodeWarrior IDE with version control systems such as SourceSafe and CVS.

Appendix A ***Perl Scripts***

Read this appendix for general information on using Perl in conjunction with the CodeWarrior IDE. It describes how to install and configure plug-in support for Perl, and describes special considerations for using Perl from the CodeWarrior IDE.

Appendix B ***CodeWarrior 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:

`typewriter`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`typewriter` (with `type` underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`typewriter italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names and buttons. Also used for terms in descriptive lists, where appropriate. Also denotes ARM processor signal names.
`typewriter 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 *ARM Developer Suite* (ADS). Refer to the following books in the ADS document suite for information on other components of ADS:

- *ADS Installation and License Management Guide* (ARM DUI 0139)
- *ADS Assembler Guide* (ARM DUI 0068)
- *Getting Started* (ARM DUI 0064)
- *ADS Compiler, Linker, and Utilities Guide* (ARM DUI 0067)
- *ADS Debuggers Guide* (ARM DUI 0066)
- *ADS Debug Target Guide* (ARM DUI 0058)
- *ADS Developer Guide* (ARM DUI 0056).

The following additional documentation is provided with the ARM Developer Suite:

- *ARM Architecture Reference Manual* (ARM DDI 0100). This is supplied in Dynatext and PDF format.
- *ARM Applications Library Programmer's Guide*. This is supplied in Dynatext and PDF format.
- *ARM ELF specification* (SWS ESPC 0003). This is supplied in PDF format in `install_directory\PDF\specs\ARMELF.pdf`.
- *TIS DWARF 2 specification*. This is supplied in PDF format in `install_directory\PDF\specs\TIS-DWARF2.pdf`.
- *ARM/Thumb Procedure Call Standard specification*. This is supplied in PDF format in `install_directory\PDF\specs\ATPCS.pdf`.

In addition, refer to the following documentation for specific information relating to ARM products:

- *ARM Reference Peripheral Specification* (ARM DDI 0062)
- the ARM datasheet or technical reference manual for your hardware device.

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 Metrowerks web site at `http://www.metrowerks.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 ARM Developer Suite and its documentation.

Feedback on the ARM Developer Suite

If you have any problems with the ARM 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 CodeWarrior for the ARM Developer Suite 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 term *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 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™. 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, 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 CodeWarrior for the ARM Developer Suite

CodeWarrior for the ARM Developer Suite is based on Metrowerks CodeWarrior IDE version 4.0. It has been tailored to support the ARM Developer Suite toolchain. It 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 most of the ARM toolchain is tightly integrated 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 ARM toolchain, such as the Runtime Settings target configuration dialog. See Chapter 9 *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. See *How the ARM debuggers work with the CodeWarrior IDE* on page 3-2 for more information.
- There are a number of menu commands and windows that are not implemented by the ARM version of the CodeWarrior IDE. The menu commands and windows not used by CodeWarrior for the ARM Developer Suite are:

  File menu

  The **Import components** menu item.

  Project menu

  The **Precompile** menu item. The ARM compilers do not support precompiled headers.

  Debug menu

  None of the menu commands in the **Debug** menu are applicable to ARM. See Chapter 3 *Working with the ARM Debuggers* for more information.

Browser menu

The following Browser menu items are not used:

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

Window menu

The following windows are not used by CodeWarrior for the ARM Developer Suite:

- Processes window
- Expressions window
- Global Variables window
- Breakpoints window
- Watchpoints window
- Register window
- Component Catalog window
- Component Palette

Help menu

The only help menu item used by CodeWarrior for the ARM Developer Suite is the **How to…** menu item.

.

Interface items that are not used by the CodeWarrior for the ARM Developer Suite are documented as *Not used by CodeWarrior for the ARM Developer Suite* 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, such as Rapid Application Development templates.

1.3 Where to go from here

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

- Read the *Getting Started* guide for a quick introduction to CodeWarrior for the ARM Developer Suite.
- Examine the example ARM projects provided in the `examples` subdirectory of your ADS installation directory.
- See Chapter 2 *Working with Projects* for detailed information on setting up your CodeWarrior projects.

  Note

  If you are setting up a complex project environment please refer to *Configuring CodeWarrior for complex or multi-user projects* on page 2-51 for important information.

- See Chapter 9 *Configuring a Build Target* for information on configuring the ARM toolchain from within the CodeWarrior IDE.

1.3.1 Online documentation and online help

Documentation for CodeWarrior for the ARM Developer Suite is available online as part of the ARM Developer Suite documentation collection. If you have installed the ADS using the default name, select **Programs** → **ARM Developer Suite v1.1** → **Online Books** from the Windows **Start** menu to access the collection.

In addition, CodeWarrior for the ARM Developer Suite provides context-sensitive online help. Select **How to…** from the **Help** menu to open the main online help. Press F1, or use the Windows help button to access context-sensitive online help.

Chapter 2
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:

2.1 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.

It also describes more complex operations, including:

- creating nested projects
- creating multiple build targets
- dividing the project window into groups of files.

In addition, it describes:

- how the CodeWarrior IDE uses project stationery
- the ARM-specific project stationery provided with this version of the CodeWarrior IDE
- how you can configure and use your own project stationery.

2.1.1 Project structure overview

A CodeWarrior 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 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 three build targets, shown in Figure 2-5 on page 2-12. See *Using ARM-supplied project stationery* on page 2-24 for more information on ARM project stationery.

When you select build options for your project you apply them specifically to one or more build targets. See Chapter 9 *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 *Creating build target dependencies* on page 2-61 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, 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 2-53 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 2-67 for more information.

2.2 Overview of the project window

The project window shows information about the files and build targets in your project file. 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*.

2.2.1 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 2-37 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 8-37 for more information on configuring toolbars in the CodeWarrior IDE.

2.2.2 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 2-1.

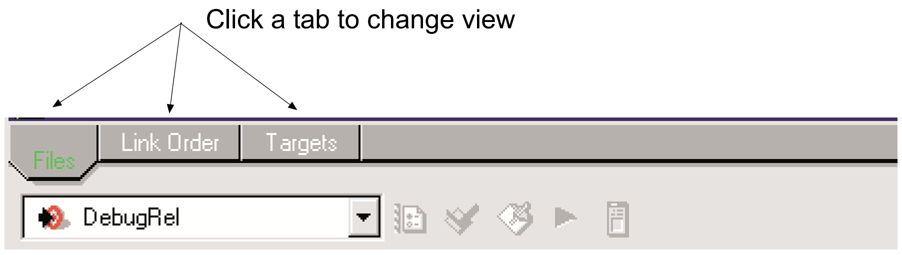


Figure 2-1 View tabs at the top of the project window

The project views are described in:

- *Files view* on page 2-5
- *Link Order view* on page 2-9
- *Targets view* on page 2-12.

Files view

The Files view (Figure 2-2 on page 2-6) 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.

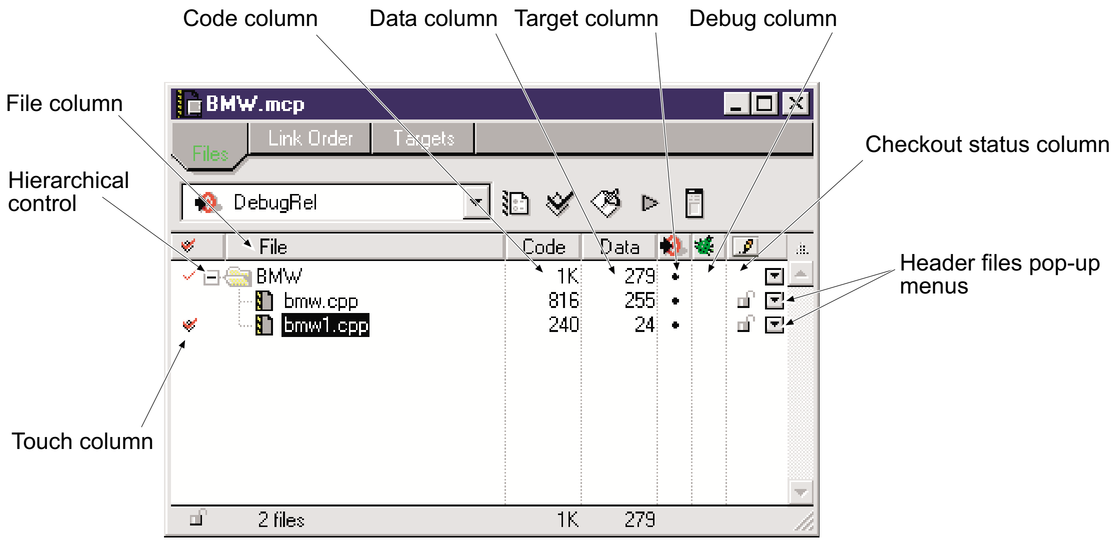


Figure 2-2 Project window Files view

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 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 pop-up 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 pop-up 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 *ADS Compiler, 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 ARM compilers 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.

See Chapter 3 *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 2-58 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 2-48 for more information.

Header Files pop-up menu

The **Header Files** pop-up 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** pop-up menu lists the files within the group. Select a file from the pop-up menu to open that file. See *Opening header files from the Header Files pop-up menu* on page 4-8 for more information.

File Control pop-up menu

The **File Control** pop-up menu is shown in Figure 2-3. To display this pop-up menu, right-click on a filename or group name in the project window.

From the **File Control** pop-up menu, you can choose a command to operate on the selected item. The available commands depend on the selected item. See Appendix B *CodeWarrior Reference* for more information on the commands in this pop-up menu.

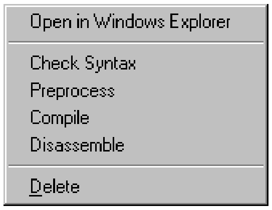


Figure 2-3 File Control pop-up menu in the project window

Checkout Status column

The Checkout Status column indicates whether files are checked in or checked out of a *Version Control System* (VCS). This column is displayed only if you configure your CodeWarrior project to use a source code revision control system. See Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information.

Project Checkout Status icon

The Project Checkout Status icon shows:

- whether a project is writable
- the file access permissions for that project.

A revision control system can assign the permissions when you check in or check out a project file. See Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information.

Link Order view

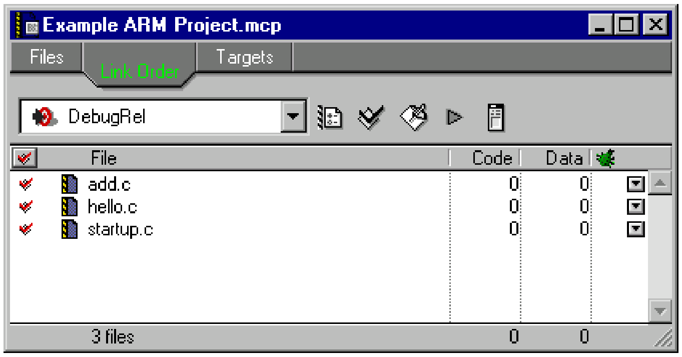
The Link Order view shows information about how the CodeWarrior IDE will compile and link the final output file for the current build target. Figure 2-4 on page 2-10 shows an example.

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 ARM linker with a list of object files in the order in which they are compiled. By default, the ARM 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 the Link Order view to control output image structure. The ARM Linker configuration panel provides limited control over section placement. For finer control, use a scatter-load description file. See *Configuring the ARM linker* on page 9-110 and the linker chapter of the *ADS Compiler, Linker, and Utilities Guide* for more information.



File	Code	Data
add.c	0	0
hello.c	0	0
startup.c	0	0
3 files	0	0

Figure 2-4 Example Link Order view

The Link Order view 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 2-5 for more information.

Touch column

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

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 2-48 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 pop-up menu* on page 4-8 for more information.

File Control pop-up menu

Right-click on an entry in the project window to display the **File Control** pop-up menu (Figure 2-3 on page 2-9 shows an example). The **File Control** pop-up menu provides context-specific commands, depending on the selected item.

Targets view

The Targets view (Figure 2-5) 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 2-5 shows an example Targets view with the three default targets defined by the ARM stationery. See *Using ARM-supplied project stationery* on page 2-24 for more information on the DebugRel, Release, and Debug build targets. See *Working with multiple build targets and subprojects* on page 2-53 for a detailed description of the Targets view, and for information on working with build targets in general.

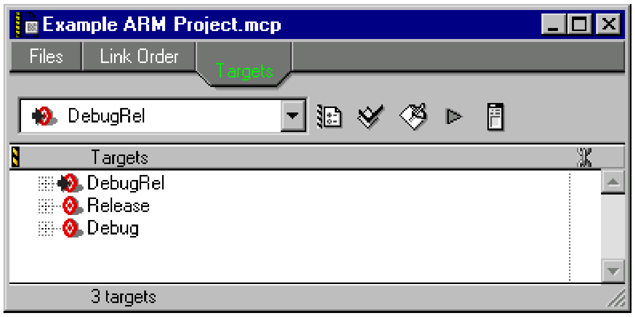


Figure 2-5 Example Targets view

2.3 Working with simple projects

This section describes the basic project operations:

- *Creating a new project*
- *Opening a project* on page 2-15
- *Closing a project* on page 2-17
- *Saving a project* on page 2-18
- *Importing makefiles into projects* on page 2-19
- *Choosing a default project* on page 2-22
- *Moving a project* on page 2-22
- *Importing and exporting a project as XML* on page 2-23.

2.3.1 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 9 *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 2-24 for more information on project stationery.

To create a new project:

1. Select **New…** from the **File** menu. The CodeWarrior IDE displays a New dialog box (Figure 2-6 on page 2-14).

   —— **Note** ——

   The New dialog box also contains a **File** tab and an **Object** tab. See *Creating a new file* on page 4-3 for more information on creating new source files. The **Object** tab is not used by the ARM version of the CodeWarrior IDE.

   ————————

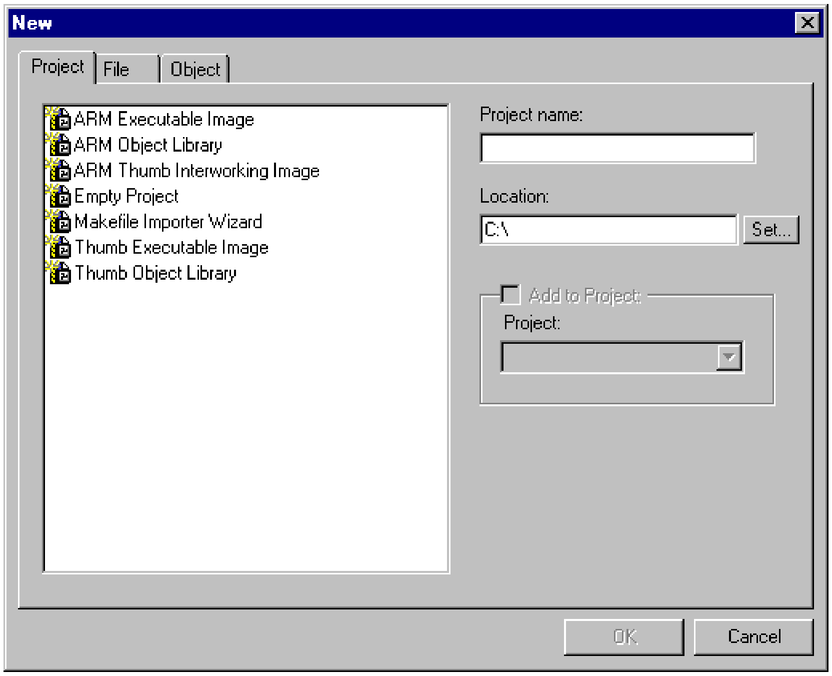


Figure 2-6 New dialog box

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 *Using ARM-supplied project stationery* on page 2-24 for more information on the project stationery files supplied by ARM.
   - The Makefile Importer Wizard. See *Importing makefiles into projects* on page 2-19 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 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 2-38, for information on how to add your own source files to the new project
   - *Compiling and linking a project* on page 2-72 for more information on building your new project.

2.3.2 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 2-22 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 2-7 on page 2-16).

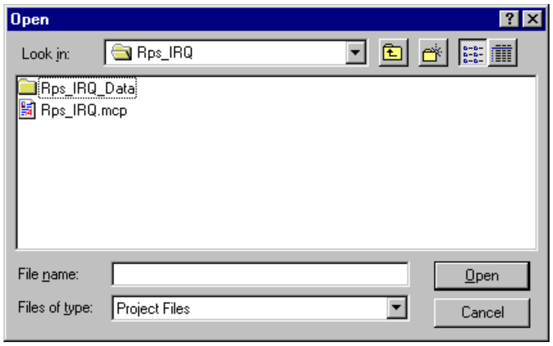


Figure 2-7 Open dialog

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

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

   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 and files you have opened recently in the **File** menu. Use the **Open Recent** menu command to reopen one of these projects. See *Configuring IDE extras* on page 8-7 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 2-53 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 MacOS 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 will be 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.

2.3.3 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 2-18 for details of how CodeWarrior saves project information.

The CodeWarrior IDE allows 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.

2.3.4 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.

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.

2.3.5 Importing makefiles into projects

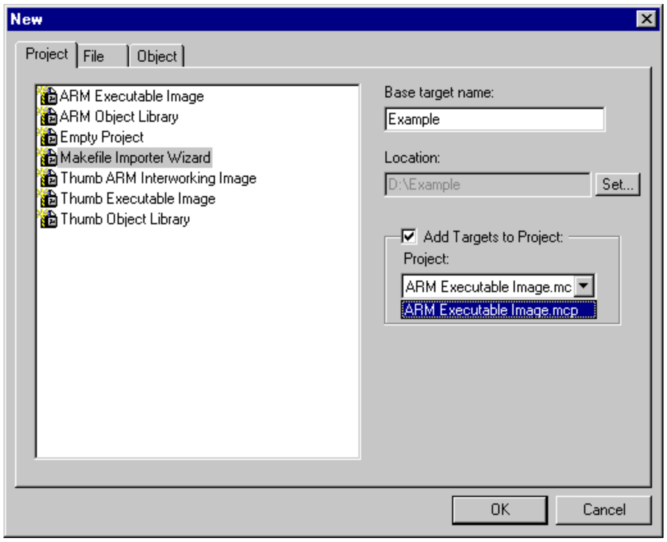
The CodeWarrior IDE can import Visual C nmake or GNU make files into CodeWarrior project files. The CodeWarrior IDE uses the Makefile Importer wizard to process the files. The wizard performs the following tasks:

- parses the makefile
- creates a CodeWarrior 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.

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 (see Figure 2-6 on page 2-14).
2. Click the **Project** tab and select the Makefile Importer Wizard from the list of project stationery.
3. Either:
    - Select **Add Targets to Project** and select a currently open project from the pop-up menu if you want to add the imported makefile to an existing project file (Figure 2-8 on page 2-20).
    - Enter a name and location for the project to be created from the imported makefile (see *Creating a new project* on page 2-13 for naming conventions) and click **OK**.



Base target name: Example

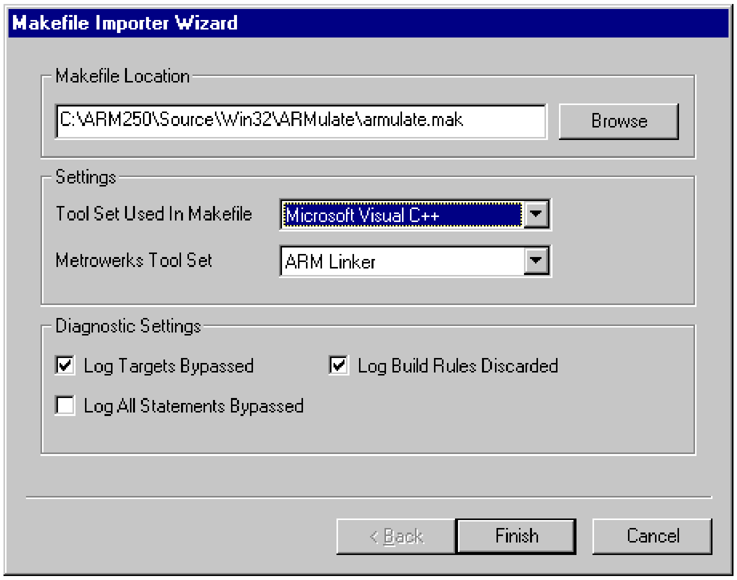
Location: D:\Example

☑ Add Targets to Project:

Project: ARM Executable Image.mcp

Figure 2-8 Adding files to an existing project

The CodeWarrior IDE displays the Makefile Importer Wizard (Figure 2-9).



Makefile Location: C:\ARM250\Source\Win32\ARMulate\armulate.mak

Settings

Tool Set Used In Makefile: Microsoft Visual C++

Metrowerks Tool Set: ARM Linker

Diagnostic Settings

☑ Log Targets Bypassed

☑ Log Build Rules Discarded

☐ Log All Statements Bypassed

Figure 2-9 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 pop-up menu.

   Metrowerks Tool Set

   Select ARM Linker from the pop-up menu.

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 build targets.

   Log Build Rules Discarded

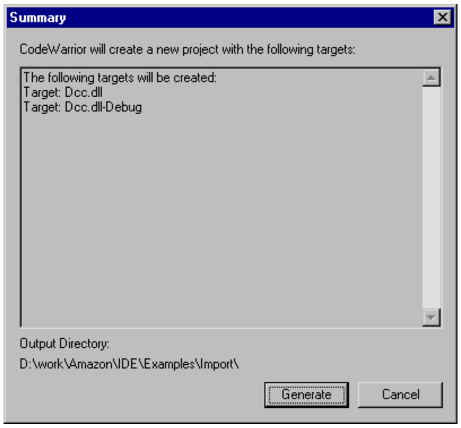
   Select this option to log information about the build rules in the makefile that were discarded in the conversion to a CodeWarrior project.

   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 3-15 for more information.

7. Click **Finish**. The Makefile Importer wizard displays a summary window showing the current conversion settings (Figure 2-10).



CodeWarrior will create a new project with the following targets:

```
The following targets will be created:
Target: Dcc.dll
Target: Dcc.dll-Debug
```

Output Directory:
D:\work\Amazon\IDE\Examples\Import\

Figure 2-10 Makefile importer summary

8. Click **Generate** to import the makefile. The CodeWarrior IDE generates a new project based on the makefile.

2.3.6 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 **Enable debugger** 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, the default project becomes the project with the front-most project window.

2.3.7 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, object code, 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. 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 9-20 for more information.

2.3.8 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 4-28 for more information.

2.4 Working with project stationery

This section describes how to use the project stationery provided with CodeWarrior for the ARM Developer Suite, and how to create your own project stationery. It describes:

- *Project stationery overview*
- *Using ARM-supplied project stationery* on page 2-24
- *Creating your own project stationery* on page 2-35.

2.4.1 Project stationery overview

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.

A stationery project can include:

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

The project stationery folder

The project stationery folder is located in the ARM Developer Suite installation folder. By default this is `c:\Program Files\ARM\ADSv1_1\Stationery`. 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. See *Creating your own project stationery* on page 2-35 for more information.

2.4.2 Using ARM-supplied project stationery

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

All the supplied stationery projects use:

- default target settings (little-endian ARM7TDMI)
- default ATPCS options for the compilers and assemblers
- separate build targets for debug and optimization options (see *Predefined build targets* on page 2-27).

This means that you must reconfigure the target options if, for example, you want to build position-independent output, or target a different ARM processor. See Chapter 9 *Configuring a Build Target* for detailed information.

The following stationery projects are supplied with the CodeWarrior IDE for the ARM Developer Suite:

ARM Executable Image

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

- The ARM C compilers to compile all files with a `.c` filename extension. If you want to use the C++ compiler to compile C code, you must reconfigure the File Mappings configuration panel. See *Configuring file mappings* on page 9-40 for more information.
- The ARM C++ compiler to compile all files with a `.cpp` filename extension.
- The ARM assembler to assemble all files with a `.s` filename extension.
- The ARM linker to link a simple executable ELF image.
- The AXD debugger to both debug and run executable images output by the project.

ARM Object Library

Use this project template to build an object library in armar format from ARM code. The library will contain ELF object format members. This project is similar to the ARM executable image project. The major differences are:

- it is configured to use the armar utility to output an object library.
- you cannot debug or run a standalone library file.

See the Toolkit Utilities chapter of the *ADS Compiler, Linker, and Utilities Guide* for more information on armar.

Thumb Executable Image

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

- The Thumb C compilers to compile all files with a `.c` filename extension. If you want to use the Thumb C++ compiler to compile C code, you must reconfigure the File mappings configuration panel. See *Configuring file mappings* on page 9-40 for more information.

- The Thumb C++ compiler to compile all files with a `.cpp` filename extension.
- The ARM assembler to assemble all files with a `.s` filename extension. By default, the Thumb Executable Image stationery configures the assembler to start in Thumb state. See the assembler documentation in the *ADS Compiler, Linker, and Utilities Guide* for information on switching the assembler to compile ARM code.
- The ARM linker to link a simple executable ELF image.
- The AXD debugger to both debug and run executable images output by the project.

Thumb Object Library

Use this project template to build an object library in armar format from Thumb code.

The library will contain ELF object format members. This project is similar to the Thumb executable image project. The major differences are:

- it is configured to use the 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 *ADS Compiler, Linker, and Utilities Guide* for more information on armar.

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 *Creating build target dependencies* on page 2-61 for information on defining build dependencies between build targets.
- The ARM C and C++ compilers to compile code included in the ARM build targets.
- The Thumb C and C++ compilers to compile code included in the Thumb build targets.
- The ARM assembler to assemble both ARM and Thumb assembly language source.
- The ARM linker to link the output from the Thumb and ARM build targets into an executable ELF image.

- The ATPCS interworking option for the ARM assembler and ARM and Thumb compilers.

Predefined build targets

The non-interworking ARM project stationery files define three build targets. The Interworking project stationery defines an additional three 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. It is intended to be used if you plan to build separate Debug and Release versions of your code. 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. It is intended to be used if you plan to build separate Debug and Release versions of your code. This build target outputs optimized code suitable for release.

DebugRel This build target is configured to build output binaries that provide adequate optimization, and give an adequate debug view. It is intended to be used if you plan to build a version of your code for debug, and release the same code in order to reduce testing.

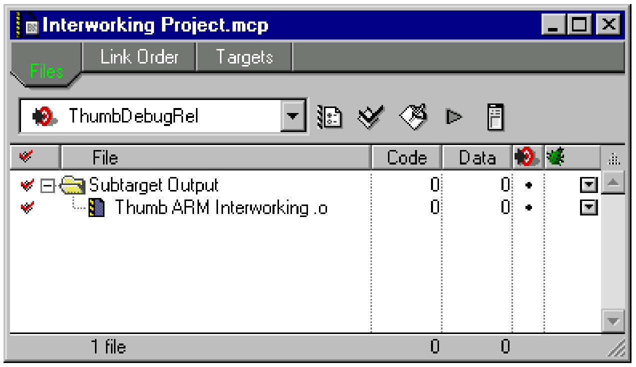
See *Working with multiple build targets and subprojects* on page 2-53 for more information on using the ARM project stationery to create complex projects. See *Configuring debug and optimization* on page 9-96 for information on how to set debug and optimization options.

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 gives a brief description of how to use the interworking stationery. See *Working with multiple build targets and subprojects* on page 2-53 for a detailed description of how to use complex projects.

To create an interworking project:

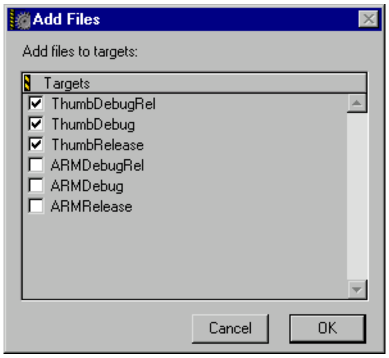
1. Create a new project using the Thumb ARM interworking stationery. See *Creating a new project* on page 2-13 for more information. The CodeWarrior IDE displays the Files view for the new project (Figure 2-11).



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

Figure 2-11 Interworking project

2. Add your Thumb source files to the Thumb build targets:
   a. Select **Add Files…** from the **Project** menu and select your Thumb source files from the standard file dialog box. The CodeWarrior IDE displays an Add Files dialog.
   b. Deselect the ARM build targets and click **OK** (Figure 2-12). The CodeWarrior IDE adds the files to the Thumb build targets only.

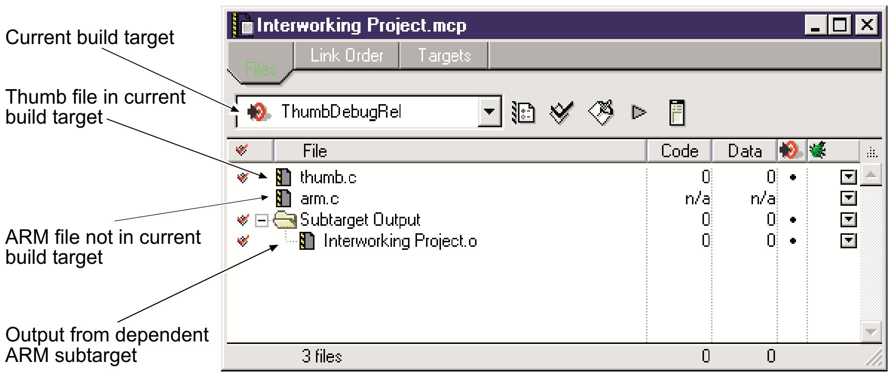


Add files to targets:

Targets
- [✓] ThumbDebugRel
- [✓] ThumbDebug
- [✓] ThumbRelease
- [ ] ARMDebugRel
- [ ] ARMDebug
- [ ] ARMRelease

Figure 2-12 Add Thumb files

3. Add your ARM source files to the ARM build targets. Follow the same procedure as step 2, but select the ARM build targets when you add the files. Figure 2-13 shows an example of the Files view for the Thumb DebugRel 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
Thumb file in current build target
ARM file not in current build target
Output from dependent ARM subtarget
Interworking Project.mcp
Files
Link Order
Targets
ThumbDebugRel
File
Code
Data
3 files

File	Code	Data
thumb.c	0	0
arm.c	n/a	n/a
Subtarget Output	0	0
Interworking Project.o	0	0
3 files	0	0

Figure 2-13 Interworking project Files view

4. Click the **Targets** tab to display the build target structure of the project. Figure 2-14 shows an example. Each ARM build target is linked as a dependent target to its corresponding Thumb build target. See *Creating a new build target* on page 2-62 for more information.

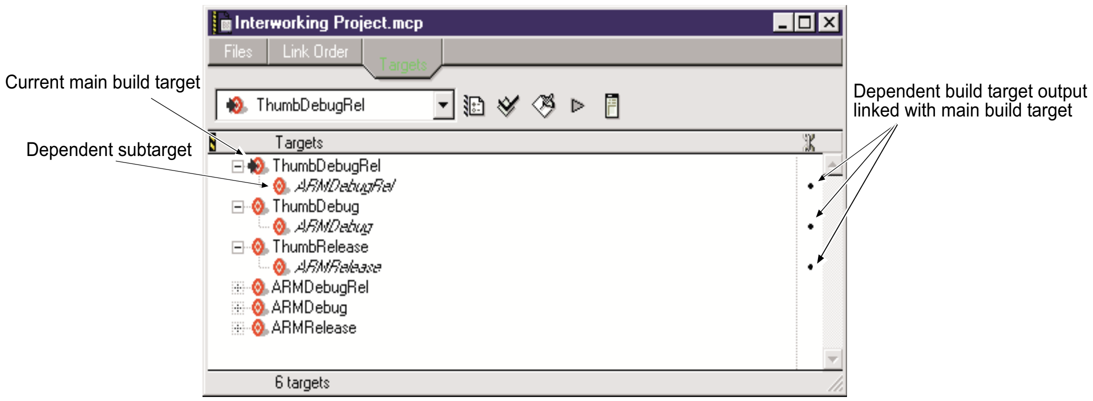


Figure 2-14 Interworking project Targets view

5. Click the **Make** button to build the ThumbDebugRel build target. The CodeWarrior IDE:
   a. builds the dependent ARMDebugRel build target
   b. builds the ThumbDebugRel build target
   c. links the output from the two build targets.

   To build the other interworking targets, select the appropriate Thumb build target from the Targets pop-up menu. If you select an ARM build target, only the ARM part of your project source is built.

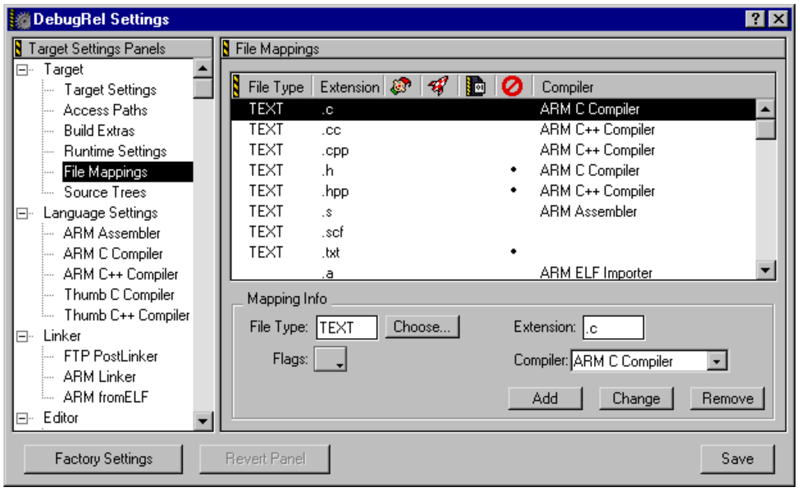
Converting ARM projects to Thumb projects

To convert an existing ARM project to Thumb, you must:

- modify the file mappings to call the Thumb compilers
- configure the appropriate ATPCS options for the compilers and assembler.

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 **File Mappings** in the Target Settings Panels list. The CodeWarrior IDE displays the File Mappings panel (Figure 2-15).



File Mappings

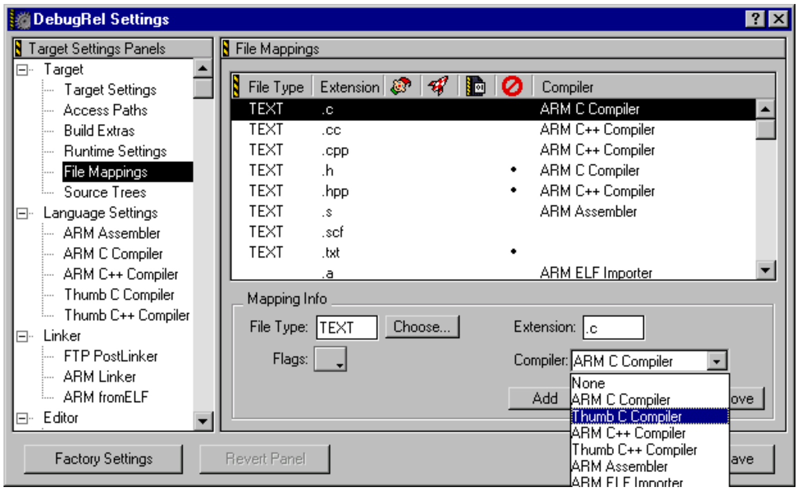
File Type	Extension					Compiler
TEXT	.c					ARM C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				•	ARM C Compiler
TEXT	.hpp				•	ARM C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf					
TEXT	.txt				•	
	.a					ARM ELF Importer

Mapping Info

File Type: TEXT
Extension: .c
Flags:
Compiler: ARM C Compiler

Figure 2-15 File Mappings panel

3. Click the entry for the `.c` file extension and select Thumb C compiler from the Compiler pop-up menu (Figure 2-16).



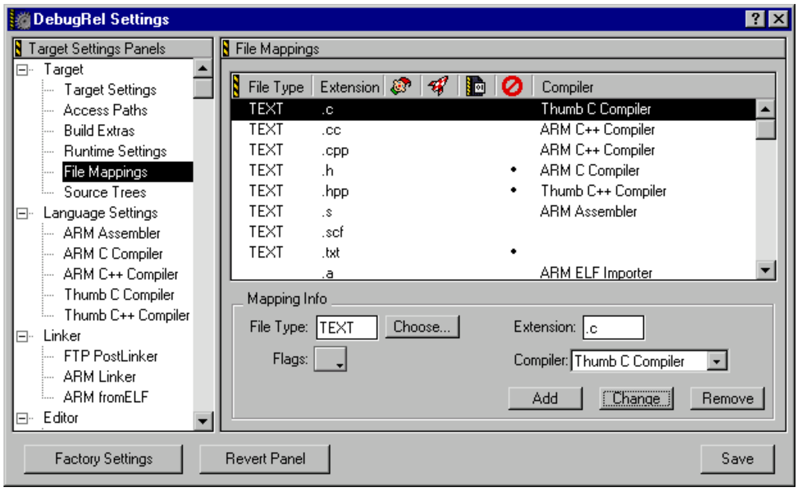
File Type	Extension					Compiler
TEXT	.c					ARM C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				•	ARM C Compiler
TEXT	.hpp				•	ARM C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf					
TEXT	.txt				•	
	.a					ARM ELF Importer

File Type: TEXT
Extension: .c
Flags:
Compiler: ARM C Compiler

Figure 2-16 Changing file mapping to use Thumb C compiler

4. Click **Change** to change the file mapping for `.c` source files. When you make your project, the CodeWarrior IDE calls the Thumb C compiler to compile files ending with a `.c` filename extension.

5. Modify the file mappings for `.cpp` and `.h` files to use the Thumb C++ compiler and Thumb C compiler respectively. The File Mappings panel should look similar to Figure 2-17.



File Type	Extension					Compiler
TEXT	.c					Thumb C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				•	ARM C Compiler
TEXT	.hpp				•	Thumb C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf					
TEXT	.txt				•	
	.a					ARM ELF Importer

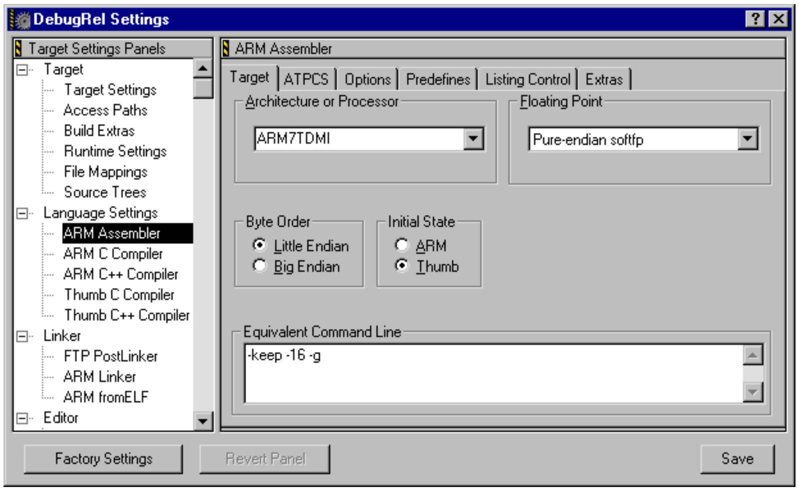
File Type: TEXT
Extension: .c
Flags:
Compiler: Thumb C Compiler

Figure 2-17 Changed file mappings

6. Click **Save** to save the new file mappings.

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

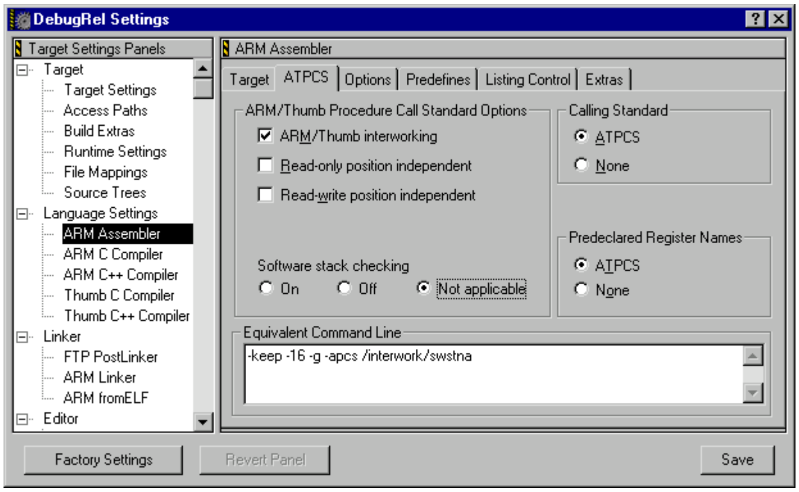
   a. Select the **Thumb** Initial State option (Figure 2-18).



Architecture or Processor: ARM7TDMI
Floating Point: Pure-endian softfp
Byte Order: [✓ Little Endian] Big Endian
Initial State: ARM [✓ Thumb]
Equivalent Command Line:
```
-keep -16 -g
```

Figure 2-18 Select Thumb initial state

   b. Click the **ATPCS** tab and select **ARM/Thumb interworking**, if required (Figure 2-19). 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.



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

Software stack checking: On Off [✓ Not applicable]
Calling Standard: [✓ ATPCS] None
Predeclared Register Names: [✓ ATPCS] None
Equivalent Command Line:
```
-keep -16 -g -apcs /interwork/swstna
```

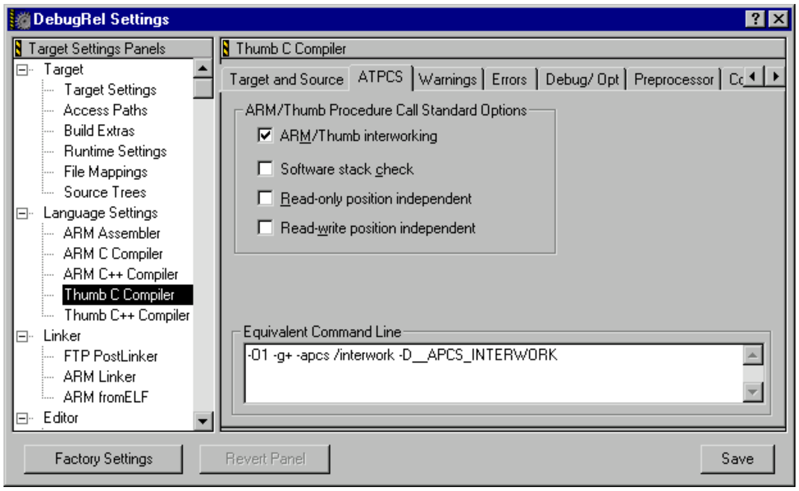
Figure 2-19 Select ARM/Thumb interworking

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

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

8. Click **Thumb C Compiler** in the Target Settings Panel list to display the Language Settings panel for the Thumb C compiler:

    a. Click the ATPCS tab and select **ARM/Thumb interworking**, if required (Figure 2-20 on page 2-33). 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.



ARM/Thumb Procedure Call Standard Options

- [✓] ARM/Thumb interworking
- [ ] Software stack check
- [ ] Read-only position independent
- [ ] Read-write position independent

Equivalent Command Line

```
-O1 -g+ -apcs /interwork -D__APCS_INTERWORK
```

Figure 2-20 Selecting the ATPCS Interworking option

    b. Ensure that the other language settings for the Thumb compiler are appropriate for your project. In particular, if you have changed any of the default settings for the ARM compiler before converting your project to Thumb, you might not have changed the equivalent settings for the Thumb compiler.

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

9. Repeat step 8 for the Thumb C++ compiler panel, if required.

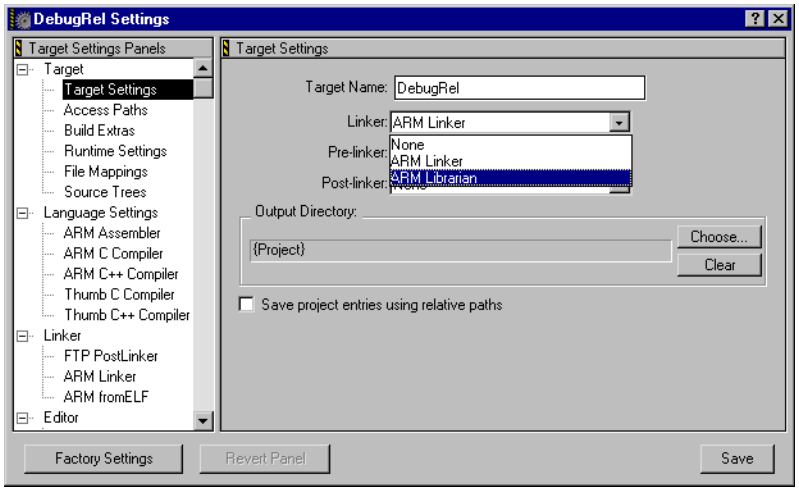
10. Repeat all the above steps for each build target in the project.

11. Rebuild your project. The CodeWarrior IDE uses the Thumb compilers to rebuild your code.

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** pop-up menu and select ARM Librarian (Figure 2-21 on page 2-34).



Target Name: DebugRel
Linker: ARM Linker
Pre-linker:
Post-linker:
Output Directory: {Project}
☐ Save project entries using relative paths

Figure 2-21 Select the ARM librarian

4. Click **Save** to save your changes.
5. Repeat all the above steps for each build target in your project.
6. Rebuild your project. The CodeWarrior IDE calls armar to create an object library.

Creating ROMable output

The default ARM-supplied project stationery is configured to generate a semihosted ELF 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 postlinker. See *Configuring target settings* on page 9-14 for more information.

2. Configure the ARM fromELF panel to convert the executable ELF image output by the linker to the binary format of your choice. See *Configuring fromELF* on page 9-129 and *Converting output ELF images to other formats* on page 2-82 for more information.

3. Configure the ARM Linker panel to create the image structure you require, or use a scatter-load description file to specify your image structure. See *Configuring the ARM linker* on page 9-110 for more information. See also the example ROM projects in `install_directory\Examples\rom` and the description of writing code for ROM in the *ADS Developer Guide*.

2.4.3 Creating your own project stationery

You can create your own stationery project file that includes the source files and project options you want. You can use the stationery project when you create a new project.

A CodeWarrior project is a stationery project if:

- it is located in the project stationery folder
- the source files associated with the project are stored with the project.

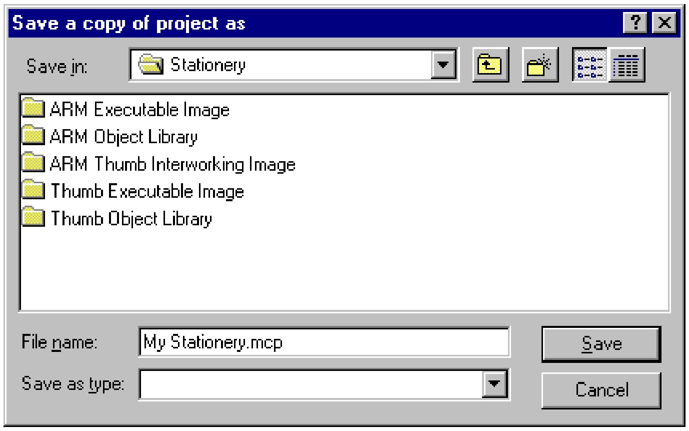
CodeWarrior duplicates the stationery project file and its source files when you create a new project and select your stationery project in the New Project dialog box. See *Creating a new project* on page 2-13 for more information on creating a new project.

Before you create your own project stationery you should be familiar with the project stationery supplied by ARM. See *Using ARM-supplied project stationery* on page 2-24 for more information.

To create your own custom stationery:

1. Create a new project from an existing project stationery file, or create an empty project.

2. Select **Save A Copy As…** from the **File** menu. The CodeWarrior IDE displays a Save a copy of project as dialog box (Figure 2-22).


Save a copy of project as

Save in: Stationery

ARM Executable Image
ARM Object Library
ARM Thumb Interworking Image
Thumb Executable Image
Thumb Object Library

File name: My Stationery.mcp

Save as type:

Figure 2-22 Save a copy of project as dialog

3. Use the dialog box controls to save the new project to the project stationery folder. By default the stationery folder is:

   `Program Files\ARM\ADSv1_1\Stationery`

4. Modify the project settings to suit your requirements. You can add and remove files as necessary to create the base project you want.

   Ensure that a copy of all the project source files are present in the project folder so that they will be copied to new projects created with the project stationery.

   ——— **Note** ———

   You do not need to copy the project data directory to the stationery folder.

   ———————————

5. Save your changes. The project is ready to use as a new project stationery file.

You can select your custom project stationery when you create a new project. The project settings and source files in your stationery project are used to create the new project.

See the following sections for more information on how to configure your project stationery and include the files you want:

- *Choosing general preferences* on page 8-6
- *Configuring target settings* on page 9-14
- *Managing files in a project* on page 2-37.

2.5 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 2-38
- *Grouping files in a project* on page 2-42
- *Moving files and groups* on page 2-45
- *Removing files and groups* on page 2-46
- *Touching and untouching files* on page 2-47.

2.5.1 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.

2.5.2 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 to add one or more files to the current project. See *Using the Add Files command* on page 2-39 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 2-41 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 2-42.

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.

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 9-40 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 2-37 for more information on selecting files and groups of files.

Note

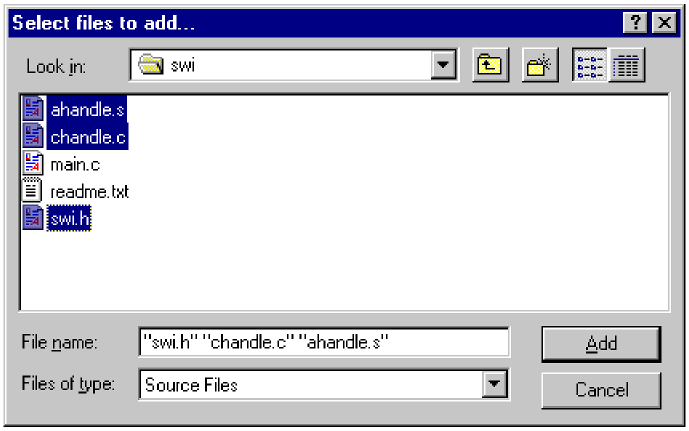
If you drag and drop a folder of source files onto your project window, a new group is created and appended to your project. The added files are placed in the new group.

See *Moving files and groups* on page 2-45 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. The CodeWarrior IDE displays an Add Files dialog box (Figure 2-23).



Select files to add...

Look in: swi

ahandle.s
chandle.c
main.c
readme.txt
swi.h

File name: "swi.h" "chandle.c" "ahandle.s"

Files of type: Source Files

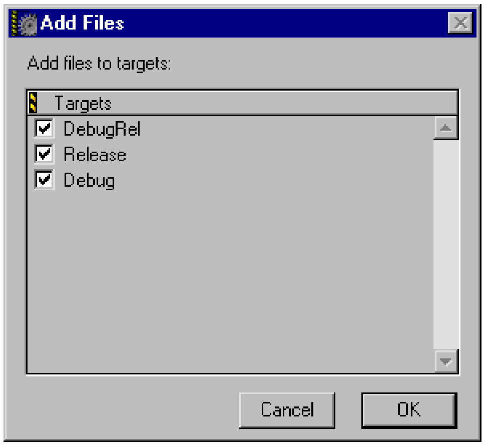
Figure 2-23 Adding files to a project

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

Note

If you select **All Files,** the dialog box displays all files regardless of their filename extension. However, the CodeWarrior IDE will not add files that do not have a recognized filename extension set in the File Mappings target configuration panel. See *Configuring file mappings* on page 9-40 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 2-24). 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.


Add Files

Add files to targets:

Targets
- [✓] DebugRel
- [✓] Release
- [✓] Debug

Figure 2-24 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 9-40 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.

   To create a new group and add files to it, drop the files when the cursor is over the blank space after the last group.

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 2-24 on page 2-40). 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 2-46 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 9-40 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 Window** 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 2-24 on page 2-40). Select the build targets to which you want the files added.

2.5.3 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 New Group** from the **Project** menu. The CodeWarrior IDE displays a Create Group dialog (Figure 2-25).

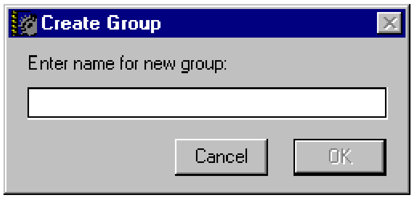


Figure 2-25 Create Group dialog

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 2-26).

   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.

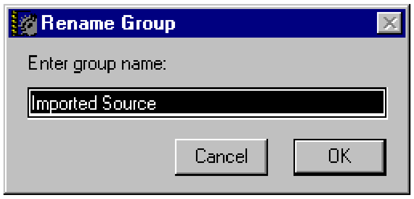

Rename Group
Enter group name: Imported Source

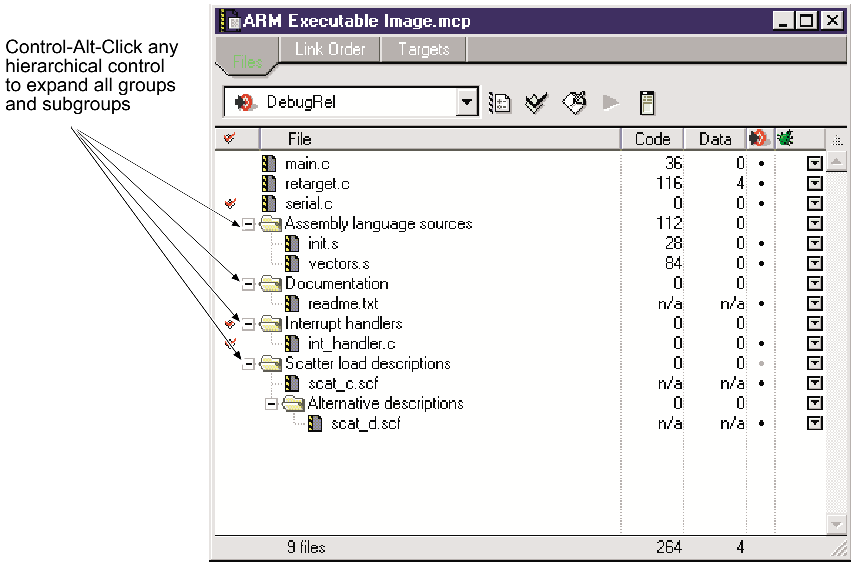
Figure 2-26 Rename group dialog

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. See Figure 2-27.


Control-Alt-Click any hierarchical control to expand all groups and subgroups
ARM Executable Image.mcp
Files
Link Order
Targets
DebugRel

File	Code	Data	
main.c	36	0	•
retarget.c	116	4	•
serial.c	0	0	•
Assembly language sources	112	0	
init.s	28	0	•
vectors.s	84	0	•
Documentation	0	0	
readme.txt	n/a	n/a	•
Interrupt handlers	0	0	
int_handler.c	0	0	•
Scatter load descriptions	0	0	•
scat_c.scf	n/a	n/a	•
Alternative descriptions	0	0	
scat_d.scf	n/a	n/a	•
9 files	264	4	

Figure 2-27 Expanding groups and subgroups

2.5.4 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 2-37 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:
   - if your selection consists only of files, the focus bar is displayed under both groups and files.
   - if your selection includes one or more groups, the focus bar is displayed only under other groups.
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 2-28 on page 2-46).

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 will be 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.

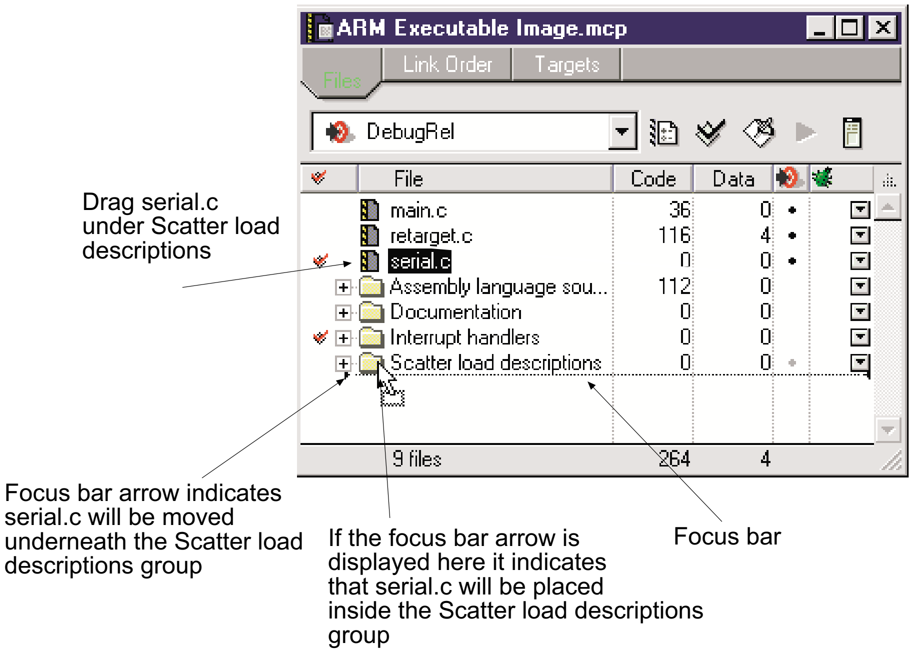

ARM Executable Image.mcp
Files
Link Order
Targets
DebugRel
File
Code
Data
main.c 36 0
retarget.c 116 4
serial.c 0 0
Assembly language sou... 112 0
Documentation 0 0
Interrupt handlers 0 0
Scatter load descriptions 0 0
9 files 264 4
Drag serial.c under Scatter load descriptions
Focus bar arrow indicates serial.c will be moved underneath the Scatter load descriptions group
If the focus bar arrow is displayed here it indicates that serial.c will be placed inside the Scatter load descriptions group
Focus bar

Figure 2-28 Moving a file

2.5.5 Removing files and groups

You can remove files from either the Files view or the Link Order view of the project window.

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 2-2 for more information on build targets.

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

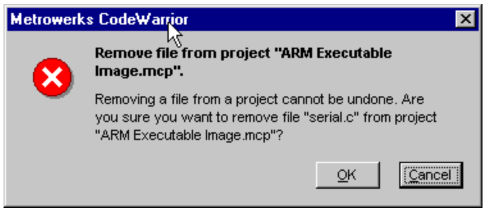
1. Click either the **Files** view tab or the **Link Order** view tab, depending on whether you want to remove the files from the entire project, or from the current build target only.
2. Select the files or groups you want to remove.

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 2-37 for more information on selecting files.

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

   The CodeWarrior IDE displays a confirmation dialog (Figure 2-29).



Metrowerks CodeWarrior

Remove file from project "ARM Executable Image.mcp".

Removing a file from a project cannot be undone. Are you sure you want to remove file "serial.c" from project "ARM Executable Image.mcp"?

Figure 2-29 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 2-38.

2.5.6 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 2-2 on page 2-6).

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 2-30).

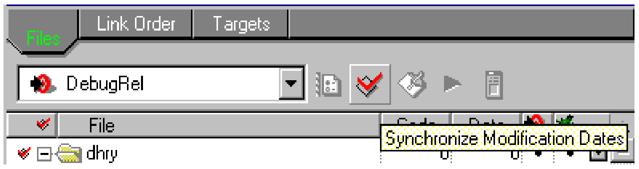


Figure 2-30 Synchronize modification dates

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.

2.5.7 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 target that include a specific file.

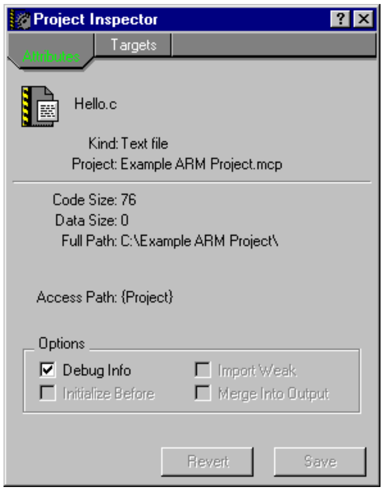
To open the Project Inspector window:

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 **Window** menu. The CodeWarrior IDE displays the Project Inspector window with the **Attributes** tab selected (Figure 2-31 on page 2-49).

   You can use this panel to specify whether debug information is generated for the current file when it is compiled. See *Controlling debugging in a project* on page 3-4 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.



Hello.c

Kind: Text file
Project: Example ARM Project.mcp

Code Size: 76
Data Size: 0
Full Path: C:\Example ARM Project\

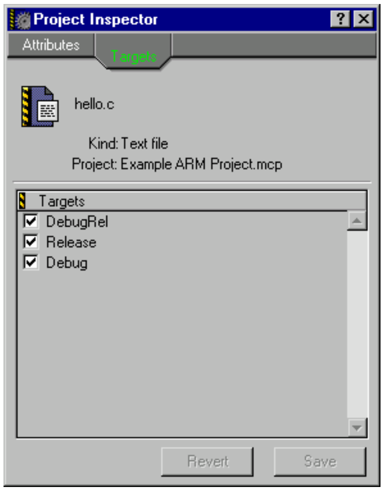
Access Path: {Project}

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

Figure 2-31 Project Inspector window for attributes

3. Click the **Targets** tab to display the Targets panel (see Figure 2-32 on page 2-50) and select the build targets that you want to include the current file.

   See *Assigning files to build targets* on page 2-58 for more information on how to select files for inclusion in specific build targets.



hello.c

Kind: Text file
Project: Example ARM Project.mcp

Targets
- [x] DebugRel
- [x] Release
- [x] Debug

Figure 2-32 Project Inspector window for targets

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

2.6 Configuring CodeWarrior 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 CodeWarrior 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 nonrecursively. However, if you make a new project, the CodeWarrior default is to add the `{Compiler}` directories recursively to your system access paths. This means that every subdirectory of the directory in which CodeWarrior is installed is added recursively to your system paths.

  See *Setting access path options* on page 9-25 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 CodeWarrior until this option is selected. In particular, if your system paths are defined to be searched recursively, CodeWarrior might find unexpected versions of source files in system paths, and ignore newer versions in user paths.

  See *Setting access path options* on page 9-25 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, CodeWarrior might not be able to determine if a source file has been modified outside the immediate

CodeWarrior 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 9-36 for more information on configuring cache modification dates option.

2.7 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
- code that is intended for release, but must still be debuggable.

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 and debugging information enabled. The release build target has code optimizations enabled. See *Using ARM-supplied project stationery* on page 2-24 for more information.

This section describes:

- *Overview of complex projects* on page 2-53
- *Creating a new build target* on page 2-57
- *Assigning files to build targets* on page 2-58
- *Changing a build target name* on page 2-60
- *Creating build target dependencies* on page 2-61
- *Building all targets in a project* on page 2-65
- *Creating subprojects within projects* on page 2-67.

2.7.1 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 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 *Creating build target dependencies* on page 2-61 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 or DebugRel build of your main project. See *Creating subprojects within projects* on page 2-67 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 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.

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 9 *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. 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 2-65 for more information.

  See *Compiling and linking a project* on page 2-72 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 pop-up menu

Select the build target from the **Build Target** pop-up menu (Figure 2-33).

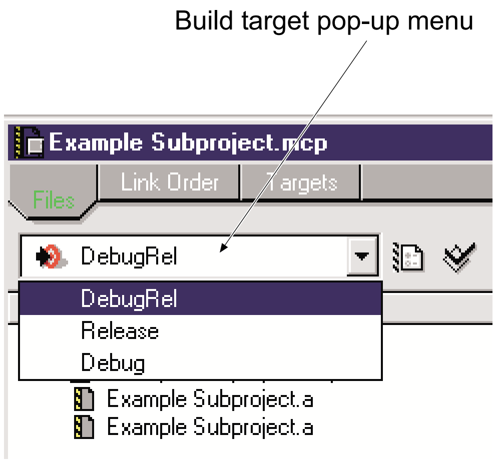


Figure 2-33 Build target pop-up menu

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 2-34 on page 2-56).

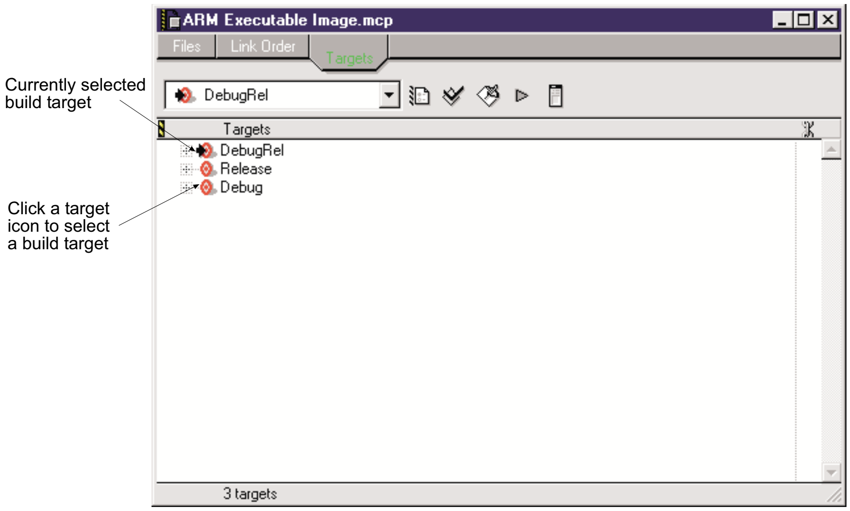


Figure 2-34 Targets view

2.7.2 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 2-35). See Figure 2-5 on page 2-12 for an example of the Targets window.

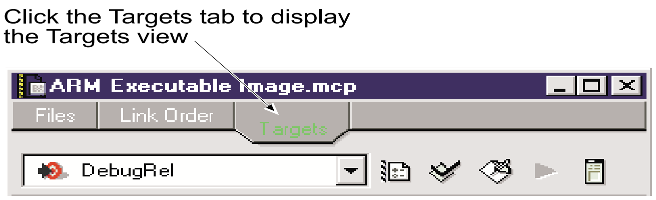
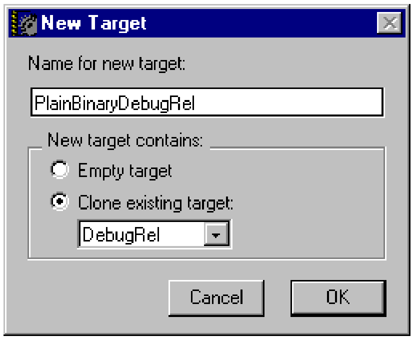


Figure 2-35 Targets view tab

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

   The CodeWarrior IDE displays a New Target dialog box (Figure 2-36 on page 2-57).



Name for new target: PlainBinaryDebugRel

New target contains:
- Empty target
- [✓ Clone existing target:] DebugRel

Figure 2-36 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* on page 2-58 for information on how to include specific source and library files in the build target.
   - Chapter 9 *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 *Creating build target dependencies* on page 2-61 for more information on defining build target dependencies.

2.7.3 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.

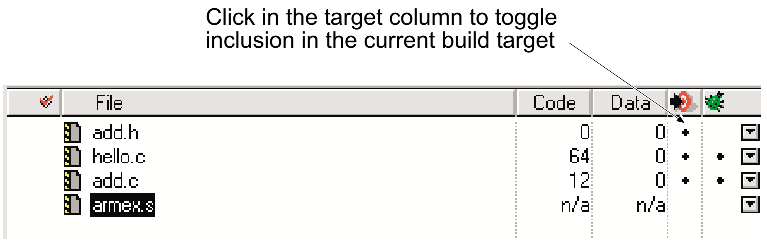
Including a file in a build target using the Target column

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 2-37 on page 2-59).

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 2-55 for information on how to change the active build target.

2. Click in the Target column next to the file (Figure 2-37):
   - 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 2-37 The target column

——— **Note** ———

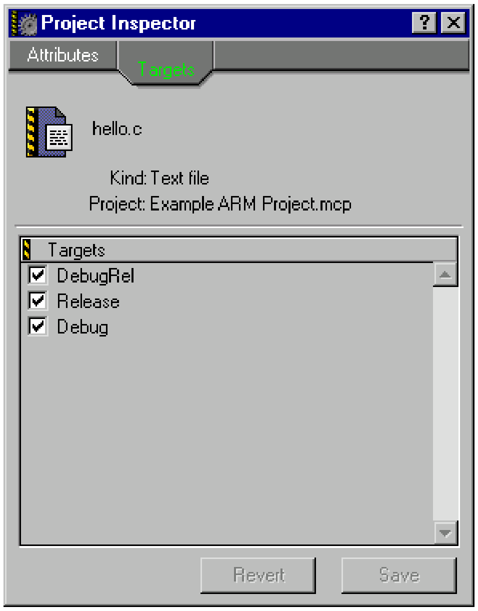
To assign or remove all the current build target files, Alt-click in the Target column.

———————————

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 **Window** 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 (see Figure 2-38 on page 2-60).



hello.c

Kind: Text file
Project: Example ARM Project.mcp

Targets
- [x] DebugRel
- [x] Release
- [x] Debug

Figure 2-38 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.

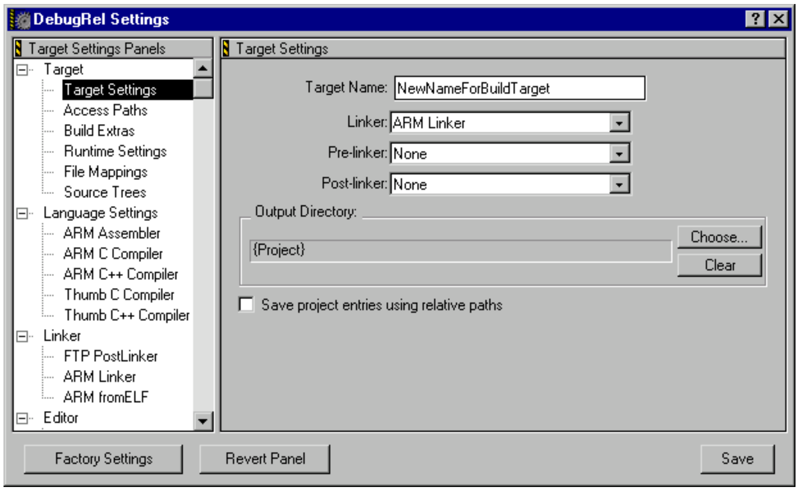
2.7.4 Changing a build target name

This section describes how to change the name of a build target. For detailed information on setting other target options see Chapter 9 *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 2-35 on page 2-57).

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 9-5 on page 9-15).

3. Select the Target Settings panel from the list of available panels and change the name of the build target in the Target Name text field (Figure 2-39).



Target Settings

Target Name: NewNameForBuildTarget

Linker: ARM Linker

Pre-linker: None

Post-linker: None

Output Directory: {Project}

☐ Save project entries using relative paths

Figure 2-39 Renaming a build target

4. Click **Save** 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 9-14 for more information.

2.7.5 Creating build target dependencies

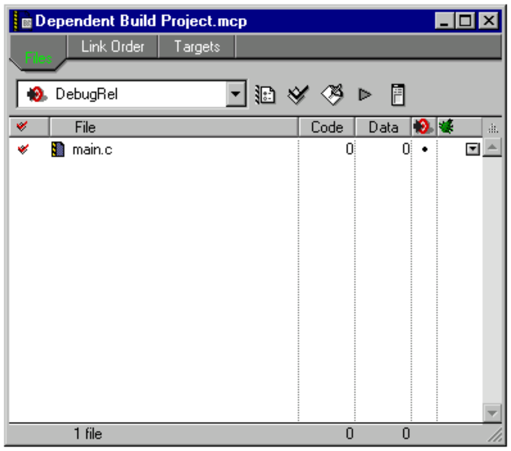
You can configure a build target to depend on other build targets. Build target dependencies are useful when you want to ensure that the CodeWarrior IDE builds one or more specific build targets before the main, containing build target. In addition, you can create link dependencies between build targets. 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 2-57 for more information on creating build targets.
- *Setting the current build target* on page 2-55 for more information on setting the current build target.
- *Strategy for creating complex projects* on page 2-54 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:

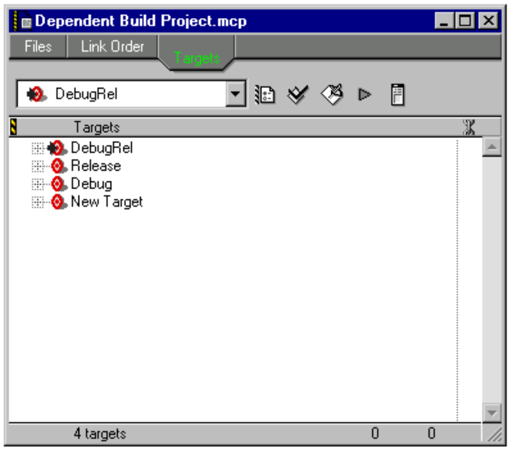
1. Open the project to which you want to add the build target. Figure 2-40 shows an example.



File	Code	Data
main.c	0	0
1 file	0	0

Figure 2-40 Project window

2. Create a new build target:
   a. Click the **Targets** tab to display the list of build targets.
   b. Select **Create New Target…** from the **Project** menu.
   c. Type a name for the new target and select whether the build target is to be based on an existing build target, or created as a new build target. See *Creating a new build target* on page 2-57 for detailed information.
   d. Click **OK**. The new Target is displayed in the list of targets (Figure 2-41). By default, the new target is not dependent on any existing targets.



Targets
DebugRel
Release
Debug
New Target

4 targets 0 0

Figure 2-41 Creating a new build target

3. 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 2-42).

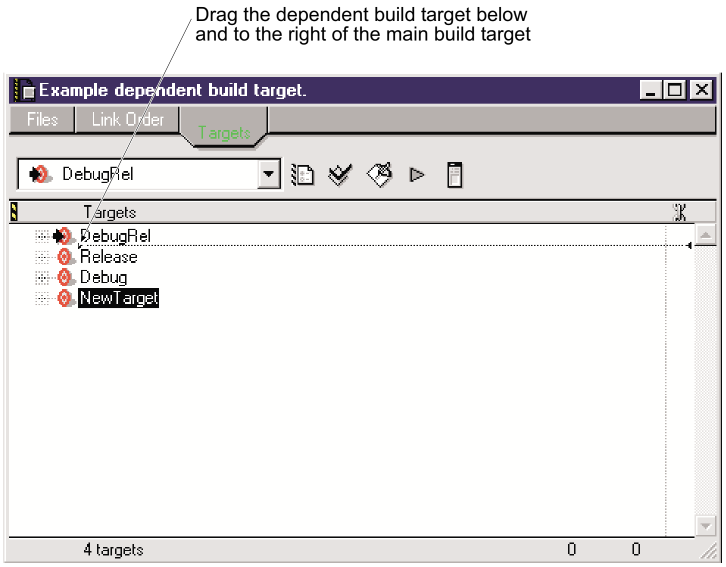

Drag the dependent build target below and to the right of the main build target
Example dependent build target.
Files
Link Order
Targets
DebugRel
Targets
DebugRel
Release
Debug
NewTarget
4 targets
0
0

Figure 2-42 Dragging a build target

4. 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 2-43 on page 2-64). 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.

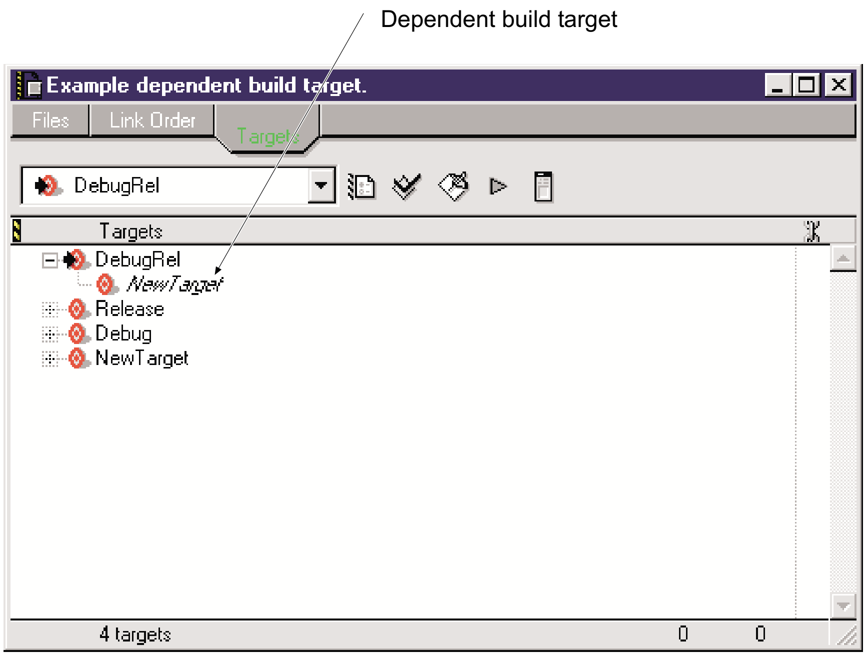
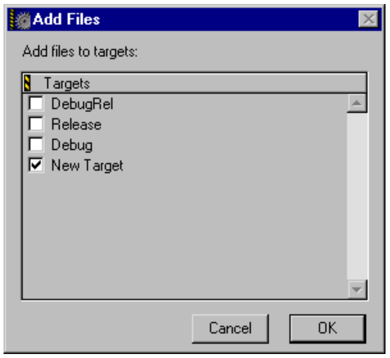


Figure 2-43 Dependent build target

5. 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 2-44).



- [ ] DebugRel
- [ ] Release
- [ ] Debug
- [x] New Target

Figure 2-44 Adding files to the dependent build target

6. (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 2-45 on page 2-65).

   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.

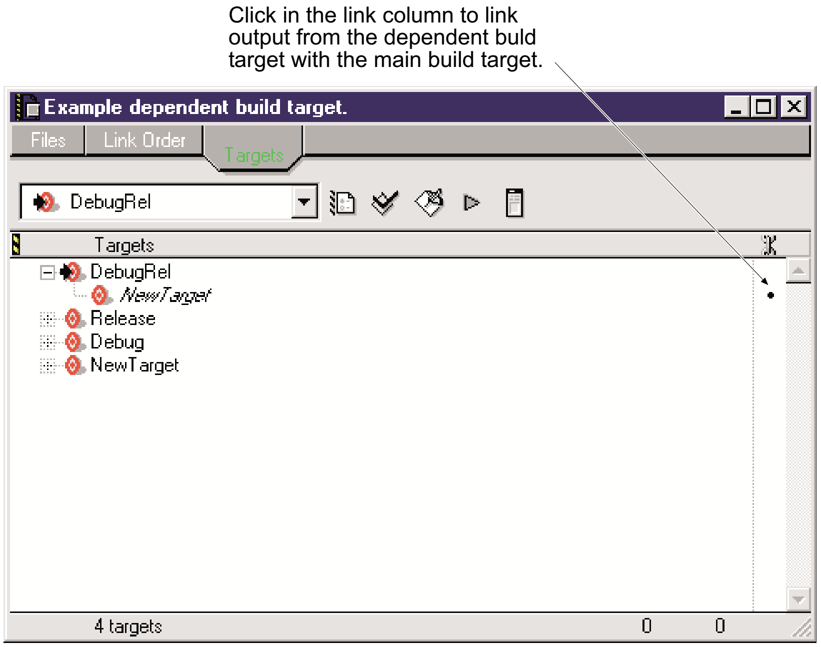

Click in the link column to link output from the dependent buld target with the main build target.
Example dependent build target.
Files
Link Order
Targets
DebugRel
Targets
DebugRel
NewTarget
Release
Debug
NewTarget
4 targets
0
0

Figure 2-45 Linking output from a dependent build target

7. 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.

2.7.6 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 2-57 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 *Creating build target dependencies* on page 2-61 for more information on creating dependent build targets.

   The Build All Targets build target displays an italicized list of dependent build targets. Figure 2-46 shows an example based on the ARM stationery. The new build target is named Build All Targets.

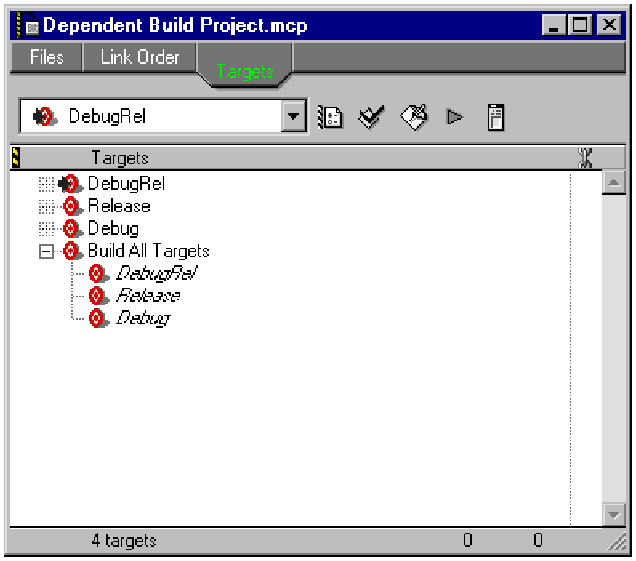


Figure 2-46 Building all build targets

4. Ensure that the Build All Targets build target is the current build target. See *Setting the current build target* on page 2-55 for more information.

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

2.7.7 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 or DebugRel build target of your main development project.

There are three important steps to compiling and linking 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 should 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 are linked when the subproject is first added.

Each of the steps is independent of the other. 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.

The following example shows how to add an ARM object library project as a subproject to an ARM executable image project, and gives details on how to specify link and build dependencies:

1. Open the project to which you want to add a subproject. In this example, the main project is an ARM Executable Image project based on the default ARM stationery. See *Using ARM-supplied project stationery* on page 2-24 for more information on the ARM stationery projects.

2. Add the subproject to the main project. In this example, the subproject is an ARM Object Library project based on the default ARM stationery. You can either:

   - Drag and drop the library project file from the Windows desktop to the main project window.
   - Select **Add Files…** from the Project menu and use the standard file dialog to select the subproject. Figure 2-47 shows an example.

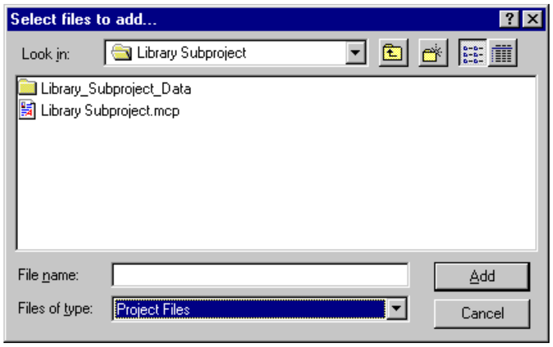


Figure 2-47 Adding a subproject

The CodeWarrior IDE displays an Add Files dialog (Figure 2-48).

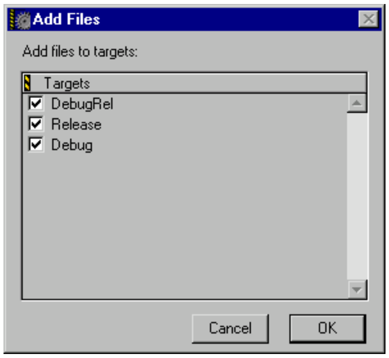
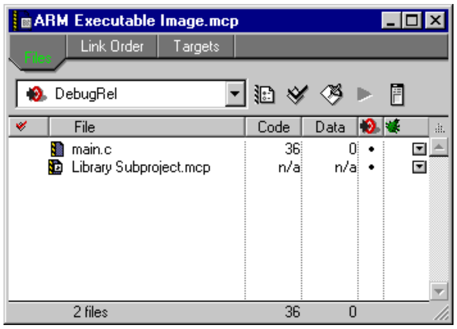


Figure 2-48 Add subproject to build target dialog

3. Select the build targets to which you want to add the library project as a subproject and click **OK**. The library project is added as a subproject to each of the selected build targets. Figure 2-49 on page 2-69 shows an example of the Files view for the project.



File	Code	Data
main.c	36	0
Library Subproject.mcp	n/a	n/a
2 files	36	0

Figure 2-49 Project with subproject

4. 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 2-50 shows an example.

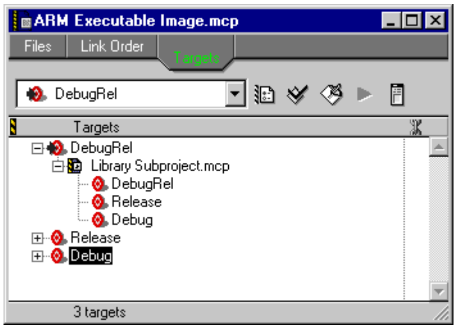


Figure 2-50 Subproject build target view

5. Click on the Target icon next to the subproject build targets you want to build when the main project is built (Figure 2-51). The CodeWarrior IDE displays an arrow and target icon for build targets that are selected. When the main project is built, selected targets are built first if they have changed, or have been touched.

   For example, in Figure 2-51 the DebugRel and Release build targets in the Example Subproject will be built before the DebugRel build target in the main project.

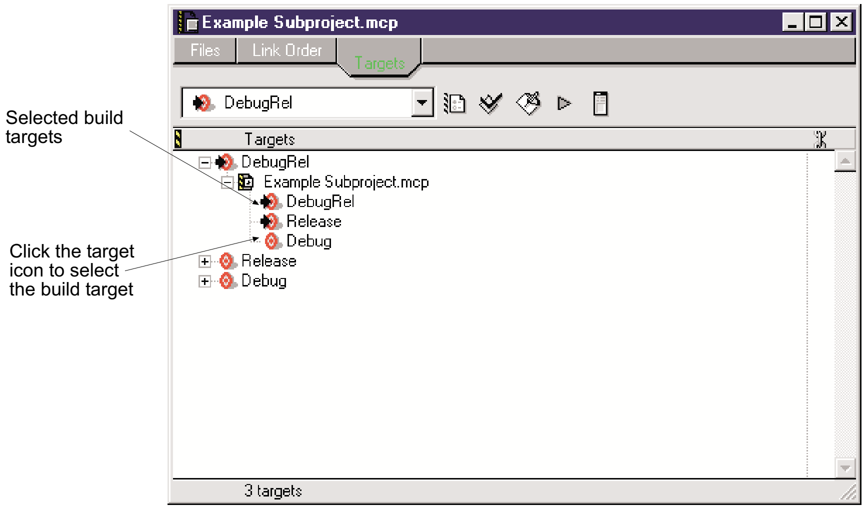


Figure 2-51 Selecting subproject build targets for building

6. Click in the link column next to the subproject build targets you want to link with the main project output (Figure 2-52 on page 2-70). You can select multiple build targets in the subproject, and link them with any of the build targets in the main project. If you select multiple subproject build targets they are linked in the order given in the link view.

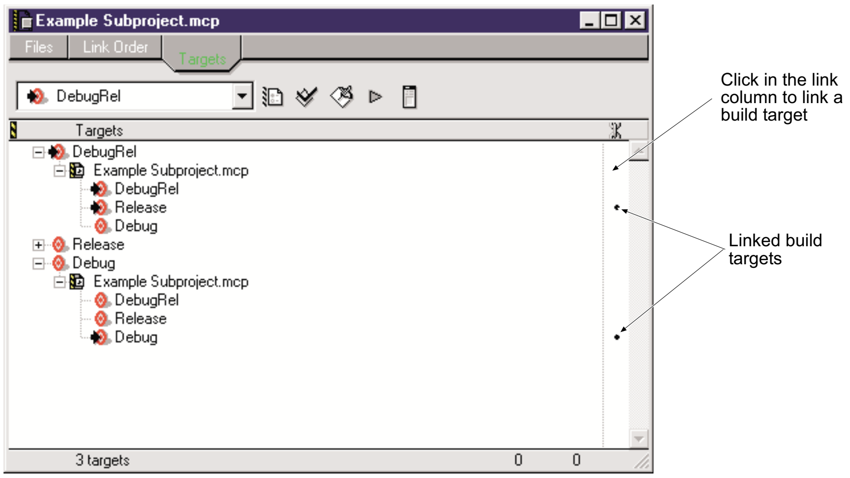
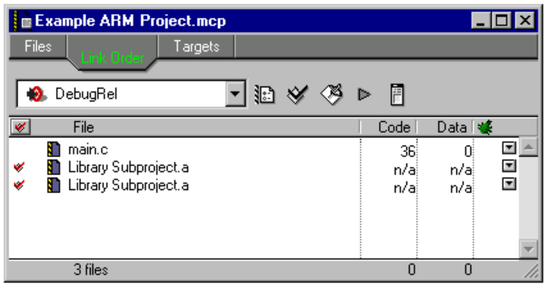


Figure 2-52 Selecting subproject build targets for linking

Caution

You can create link dependencies to any of the build targets in a subproject. This means that, if you are using the ARM-supplied project stationery, you can create a link dependency to any, or all, of the Debug, DebugRel, and Release build targets in the subproject, all of which might contain the same code, built with different optimization and debug options.

When you build your project, the ARM linker selects the first available object file that resolves the unresolved symbol it is processing. In the Link order view, the output filenames for each of the build targets are identical (see Figure 2-53 on page 2-71). Select the output file in the Files view and use the Project Inspector to determine which output object is being linked. See *Examining and changing project information for a file* on page 2-48 for more information.



	File	Code	Data
	main.c	36	0
✓	Library Subproject.a	n/a	n/a
✓	Library Subproject.a	n/a	n/a
	3 files	0	0

Figure 2-53 Multiple build targets in the Link order view

2.8 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 CodeWarrior is executing another command.

If you have multiple projects open at the same time, you can set the default project that CodeWarrior will use. See *Choosing a default project* on page 2-22 for more information.

This section describes:

- *Overview of compiling and linking*
- *Compiling files* on page 2-74
- *Making a project* on page 2-77
- *Removing objects from a project* on page 2-79.

2.8.1 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 should also be familiar with features such as moving files in the project window, the project window columns, and project window pop-up menus. See *Overview of the project window* on page 2-4 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.

Choosing a compiler

The CodeWarrior IDE uses file mappings to associate a compiler, or other tool, with a specific filename extension. For C and C++ source files, the CodeWarrior IDE uses the file mappings for the current build target to distinguish between source files targeted at the Thumb C compiler and the ARM C compiler.

In the project stationery provided with CodeWarrior for the ARM Developer Suite:

- the ARM project stationery maps `.c` files to the ARM C compiler
- the Thumb project stationery maps `.c` files to the Thumb C compiler.

To change the compiler used for a build target, you must change the file mapping for that build target. See *Using ARM-supplied project stationery* on page 2-24 and *Configuring file mappings* on page 9-40 for more information.

If you want to select between the Thumb compiler and ARM compiler for source files in the same build target, you can adopt your own file naming conventions. For example, to identify Thumb source files:

- Use a filename extension such as `.tcc` for all source files that you want to compile with the Thumb compiler
- Define a file mapping between the Thumb compiler and `.tcc` files in the Target configuration panels for the build target. See *Configuring file mappings* on page 9-40 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 DebugRel build target and you recompile your source files, the object code for the Release and Debug build targets 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 2-55 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 *Creating build target dependencies* on page 2-61 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 9-14 for more information. Table 2-1 describes the output file naming conventions and default locations.

Table 2-1 Default output names and locations

Output	Naming convention	Default location in the project folder
Executable ELF image	`Project Name.axf`	`Project_Name_Data\Target_Name`
Partially linked ELF object	`Project Name.o`	`Project_Name_Data\Target_Name`
ARM library	`Project Name.a`	`Project_Name_Data\Target_Name`
Object code	`filename.o`	`Project_Name_Data\Target_Name\ObjectCode`

2.8.2 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.

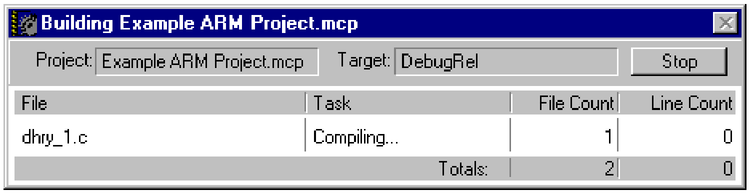
The object files generated by the compilation are placed in a data subdirectory of your main project folder. See *Making a project* on page 2-77 for information on compiling and linking your source files and libraries.

Note

The CodeWarrior IDE compiles the files only for the currently selected build target. See *Setting the current build target* on page 2-55 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 2-54). The Build Progress window displays a line count and the name of the file currently being compiled.



Project: Example ARM Project.mcp Target: DebugRel

File	Task	File Count	Line Count
dhry_1.c	Compiling...	1	0
	Totals:	2	0

Figure 2-54 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:

   - there is no open project
   - 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 2-37 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 2-55 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 CodeWarrior 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 2-47 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 2-48 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 `Preprocessed` *`filename`*.
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 messages window.

2.8.3 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

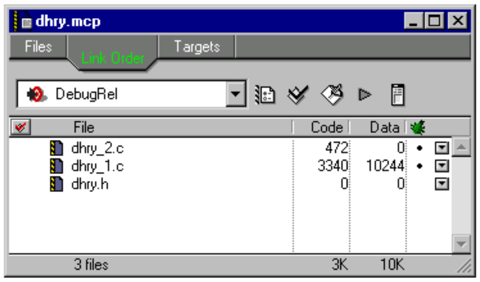
- linking object files and libraries to produce an ELF image file, or a partially linked object
- performing any postlink operations that you have defined for your build target, such as calling fromELF to convert an ELF image file to another format.

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 2-55).



File	Code	Data
dhry_2.c	472	0
dhry_1.c	3340	10244
dhry.h	0	0
3 files	3K	10K

Figure 2-55 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 2-9 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 ARM linker with a list of object files in the order in which they are compiled.

  By default, the ARM 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 ARM linker* on page 9-110 and the linker chapter of the *ADS Compiler, Linker, and Utilities Guide* for more information.

- You can generate link information by selecting options in the ARM Linker configuration panel and remaking your project. See *Configuring the ARM linker* on page 9-110 for more information.

2.8.4 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 project `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

Do not delete the contents of the `data` directory manually.

Removing object code

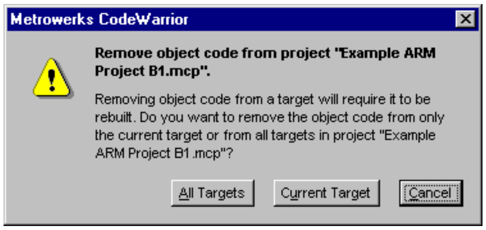
To remove the object code from a project:

1. Ensure that the project window for the project is the current window, or that the project from which you want to remove object code is selected as the current default project.

   Note

   If you remove object code while an editor window is active, the CodeWarrior IDE 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 a Remove Objects dialog box (Figure 2-56).



Remove object code from project "Example ARM Project B1.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 "Example ARM Project B1.mcp"?

Figure 2-56 Remove Objects dialog box

3. 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.
    - **Current Target**, to remove the objects for the current build target only. See *Setting the current build target* on page 2-55 for more information on changing the build target.
    - **Cancel**, to cancel the operation so that object code is not removed.

2.9 Processing output

This section describes how to process project output. It describes:

- *Disassembling code*
- *Converting output ELF images to other formats* on page 2-82
- *Creating libraries with armar* on page 2-83
- *Running batch files with the batch runner* on page 2-84.

2.9.1 Disassembling code

You can configure the CodeWarrior IDE to call the ARM command-line tool fromELF to display an assembly language listing for an object file, or library file. 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 2-83 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 9-14 and *Configuring fromELF* on page 9-129 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 pop-up 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.

2.9.2 Converting output ELF images to other formats

You can configure the CodeWarrior IDE to call 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
- Extended Intellec Hex
- Verilog Hex Format.

See the *ADS Compiler, 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 9-14 for detailed instructions.
2. Configure the fromELF utility to generate the output you want. See *Configuring fromELF* on page 9-129 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\ProjectName_Data\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 9-14 for detailed instructions.
2. Configure the fromELF utility to generate the output you want. See *Configuring fromELF* on page 9-129 for detailed instructions.
3. Configure the ARM Debugger or the ARM Runner panel to call fromELF as a third-party debugger to disassemble the converted output binary. See *Configuring the ARM Debugger* on page 9-136 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 ARM Runner* on page 9-153 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 *ADS Compiler, Linker, and Utilities Guide* for detailed information on the command-line options to fromELF.

2.9.3 Creating libraries with armar

To configure the CodeWarrior IDE to call armar to output libraries in ar format you must configure the Target Settings panel to call armar as the linker. See *Configuring target settings* on page 9-14 for more information. armar 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 *Using ARM-supplied project stationery* on page 2-24 for more information.

2.9.4 Running batch files with the batch runner

The CodeWarrior IDE 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 is not configured to recognize batch files.
- Configure the batch runner as the postlinker in the Target Settings panel.

  Note

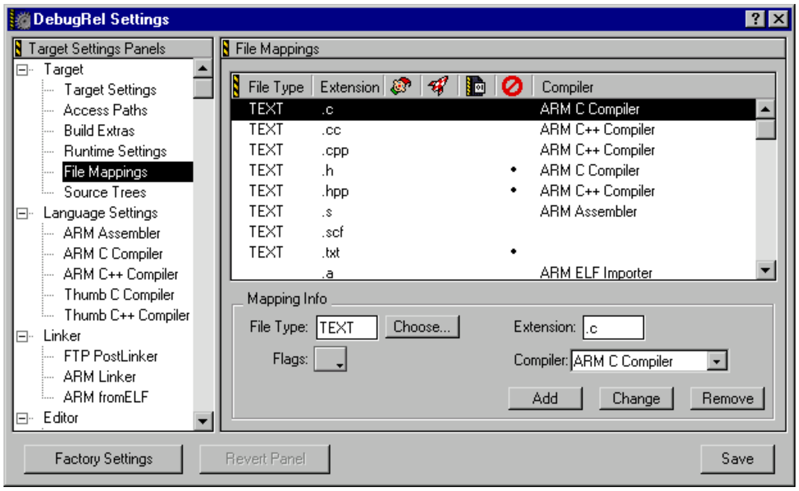
  You can configure only one postlinker. This means that you cannot use the batch runner and fromELF in the same build target.

- Add the batch file to the build target and make the build target. The batch runner is run only after a successful link operation.

Configuring file mappings to recognize batch files

You must configure the file mappings for each build target to which you want to add batch files.

1. Display the Target Settings panel for the build target you want to configure (see *Displaying Target Settings panels* on page 9-8).
2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel (Figure 2-57 on page 2-85).



File Type	Extension					Compiler
TEXT	.c					ARM C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				•	ARM C Compiler
TEXT	.hpp				•	ARM C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf					
TEXT	.txt				•	
	.a					ARM ELF Importer

Mapping Info

File Type: TEXT Choose... Extension: .c

Flags: Compiler: ARM C Compiler

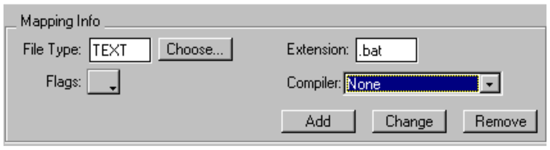
Add Change Remove

Figure 2-57 File Mappings 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** pop-up menu and select **None**. Figure 2-58 shows an example.



Mapping Info

File Type: TEXT Choose... Extension: .bat

Flags: Compiler: None

Add Change Remove

Figure 2-58 Specifying batch file mappings

6. Click **Add** to add a new file mapping.

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

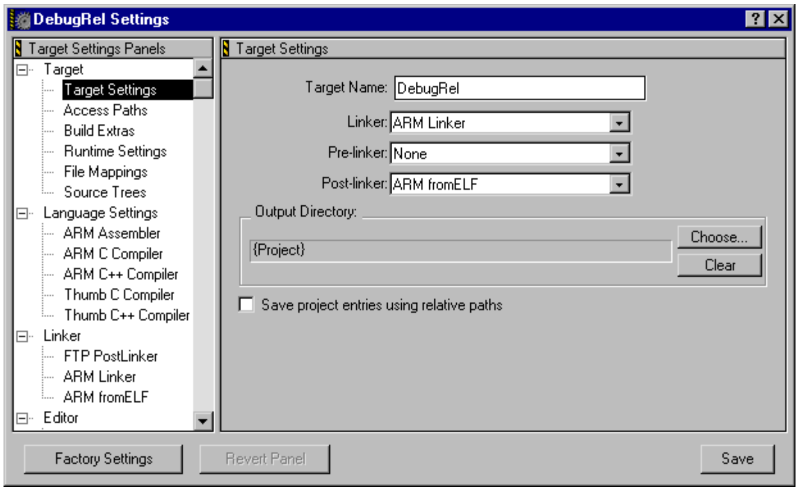
Configuring the batch runner as the postlinker

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 *Displaying Target Settings panels* on page 9-8).

2. Click **Target Settings** in the Target Settings Panels list to display the configuration panel (Figure 2-59).



DebugRel Settings

Target Settings

Target Name: DebugRel

Linker: ARM Linker

Pre-linker: None

Post-linker: ARM fromELF

Output Directory: {Project}

☐ Save project entries using relative paths

Figure 2-59 Target Settings panel

3. Click the **Post-linker** pop-up menu and select **Batch File Runner**. See *Configuring target settings* on page 9-14 for more information on the other options on this panel.

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

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 IDE displays a warning message and adds the batch files only to the properly configured targets.

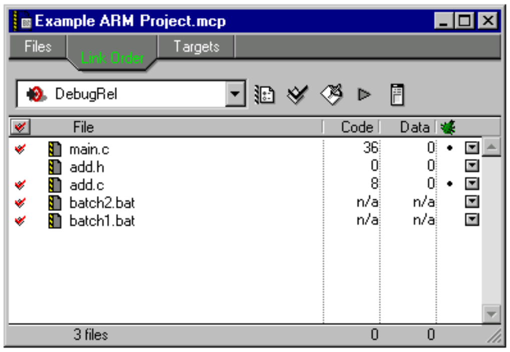
To add the batch files and make your build target:

1. Either:
   - Select **Add Files…** from the **Project** menu
   - Drag and Drop the batch files onto the project window.

See *Adding files to a project* on page 2-38 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 link step. If your project is up to date, the linker is not executed and the batch file runner is not run. See *Touching and untouching files* on page 2-47 for more information.

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 2-60).



✓	File	Code	Data	
✓	main.c	36	0	•
	add.h	0	0	
✓	add.c	8	0	•
✓	batch2.bat	n/a	n/a	
✓	batch1.bat	n/a	n/a	
	3 files	0	0	

Figure 2-60 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, `batch2.bat` in Figure 2-60 is executed first.

Chapter 3
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:

3.1 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.

3.1.1 How the ARM debuggers work with the CodeWarrior IDE

CodeWarrior for the ARM Developer Suite 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 CodeWarrior for the ARM Developer Suite.

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:

- *ARM eXtended Debugger* (AXD)
- *ARM Debugger for Windows* (ADW)
- *ARM symbolic debugger* (armsd).

In addition, you can select a third party debugger in place of the ARM debuggers. See *Configuring the debugger* on page 9-135 for detailed information on selecting a debugger and a runner. See the *ADS Debuggers Guide* for detailed information on using the ARM debuggers.

Using the Run/Debug button

The CodeWarrior IDE uses the same button both to run and to debug output images, depending on whether or not debugging is enabled for your project. The tool tip for the button displays either *Run* or *Debug*, depending on how the build target is configured. See *Enabling debugging for a build target* on page 3-4 for more information.

Selecting debug options

There are a number of options to the ARM compilers 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, DebugRel, and Release build targets. See below for more information. See *Configuring the compilers* on page 9-72 for more information on setting build options yourself.

Using the Debug, DebugRel, and Release build targets

The default project stationery provided with CodeWarrior for the ARM Developer Suite defines at least three 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.
DebugRel	This build target is configured to generate adequate debug information for each source file in the build target.
Release	This build target is configured with debug table generation turned off.

Note

You must select **Enable Debugger** for the Debug and DebugRel build targets in order to debug, rather than run, your output image. See *Enabling debugging for a build target* on page 3-4 for more information.

See *Using ARM-supplied project stationery* on page 2-24 for more information on the default build targets. See also *Working with multiple build targets and subprojects* on page 2-53 for more information on using multiple build targets in your projects.

3.2 Controlling debugging in a project

You can specify whether debugging is enabled for one or more source files, for each build target in a project. To enable debugging for a build target:

- Select **Enable Debugger** from the **Project** menu to instruct the CodeWarrior IDE to execute a debugging session, rather than run session, when you click the **Run/Debug** button in the project toolbar. In some circumstances, this command also turns on debug table generation for all source files in the build target. See *Enabling debugging for a build target* on page 3-4 for more information.
- Configure the ARM tools to generate debug table information when your source code is compiled and assembled. There are two, independent 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* on page 3-5 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 3-7 for more information.

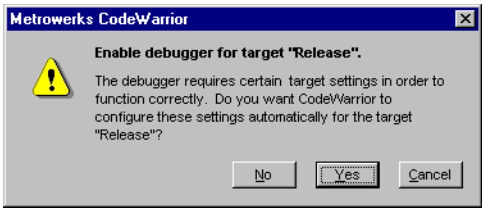
Note

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.

3.2.1 Enabling debugging for a build target

To enable debugging for the current build target:

1. Select the build target for which you want to enable debugging. See *Setting the current build target* on page 2-55 for more information.
2. Select **Enable Debugger** from the **Project** menu. The CodeWarrior IDE response depends on whether or not source files in the current build target have debugging enabled in debug column:
   - If no source files have debugging enabled in the debug column, the CodeWarrior IDE displays a confirmation dialog (Figure 3-1 on page 3-5). See the steps below for more information.



Metrowerks CodeWarrior

Enable debugger for target "Release".

The debugger requires certain target settings in order to function correctly. Do you want CodeWarrior to configure these settings automatically for the target "Release"?

Figure 3-1 Enable debugging confirmation

- If at least one source file has debugging enabled in the debug column, the CodeWarrior IDE does not change the debug status of any source files in the build target. The debug target is configured to execute a debug session when you click the **Debug/Run** button in the project toolbar.

3. Click **Yes** if the Enable debugging confirmation dialog is displayed and you want to enable debugging. The CodeWarrior IDE:
   - enables debugging in the debug column for every source file in the current build target
   - is configured to execute a debug session when you click the **Run/Debug** button in the project toolbar.

   Caution

   Selecting **Disable Debugger** from the **Project** menu does not deselect debugging for individual source files. This means that, if your project is based on the default ARM stationery and you turn on debugging for the Release build target, you must manually deselect the source files in order to turn debug table generation off again, and build an output image without debug tables. See *Generating debug information for individual source files* on page 3-5 for more information.

Adding source files to a debug-enabled build target

When you add source files to a build target, the debug state of the file is configured to match the state of the **Enable Debugger** menu command setting in the **Project** menu.

3.2.2 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 3-7 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 2-48 for more information.

To generate debugging information for a source code file:

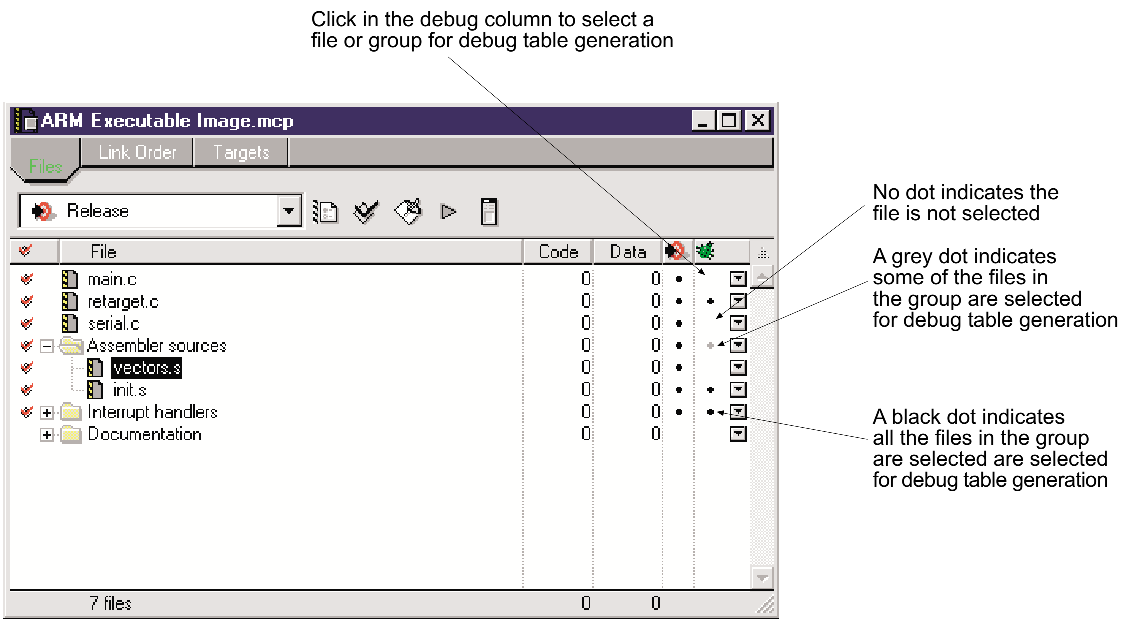
1. Select the build target for which you want to enable debugging. See *Setting the current build target* on page 2-55 for more information.
2. Select the source files or groups for which you want to enable debugging:
   - Click in the Debug column next to a single source file to turn on debug table 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 enable debugging 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 (on page 3-7) 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

No dot indicates the file is not selected

A grey dot indicates some of the files in the group are selected for debug table generation

A black dot indicates all the files in the group are selected are selected for debug table generation

✓	File	Code	Data		
✓	main.c	0	0	•	
✓	retarget.c	0	0	•	•
✓	serial.c	0	0	•	
✓	Assembler sources	0	0	•	◦
✓	vectors.s	0	0	•	
✓	init.s	0	0	•	•
✓	Interrupt handlers	0	0	•	•
	Documentation	0	0		
	7 files	0	0		

Figure 3-2 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.

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

There are two ways in which you can configure the CodeWarrior IDE to generate debug information for all files in the current build target:

- Select **Enable Debugger** from the **Project** menu when no files in the current build target are individually selected to generate debug information:
  - 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 *Enabling debugging for a build target* on page 3-4 for more information.

- Use the ARM compiler and 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 each compiler, and for the ARM assembler.

Note

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

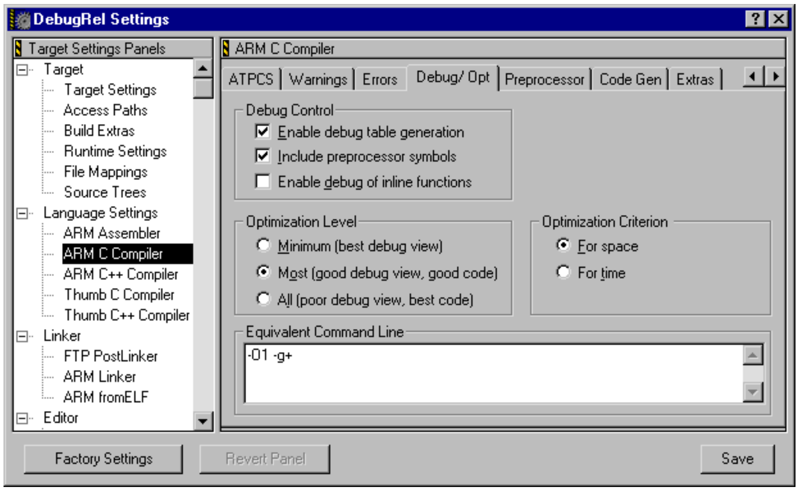
Example: Using the compiler configuration dialogs

The following example shows how to enable debug setting for the ARM C compiler. The steps for the assembler and C++ compilers are 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 2-73 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 9-8 for more information.

3. Click the ARM C Compiler entry in the Target Settings Panels list, and click the **Debug/Opt** tab to display the configuration panel (Figure 3-3 on page 3-8).



Debug Control
- [✓] Enable debug table generation
- [✓] Include preprocessor symbols
- [ ] Enable debug of inline functions

Optimization Level
- ○ Minimum (best debug view)
- ● Most (good debug view, good code)
- ○ All (poor debug view, best code)

Optimization Criterion
- ● For space
- ○ For time

Equivalent Command Line

```
-O1 -g+
```

Figure 3-3 ARM 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 9-96 for more information on the other options available on this panel. See *Configuring assembler options* on page 9-63 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.

3.3 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.

3.3.1 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. Ensure that debugging is *not* activated for your project. See *Enabling debugging for a build target* on page 3-4 for more information.
3. Select the **Run** from the **Project** menu, or click the **Run/Debug** button (Figure 3-4).

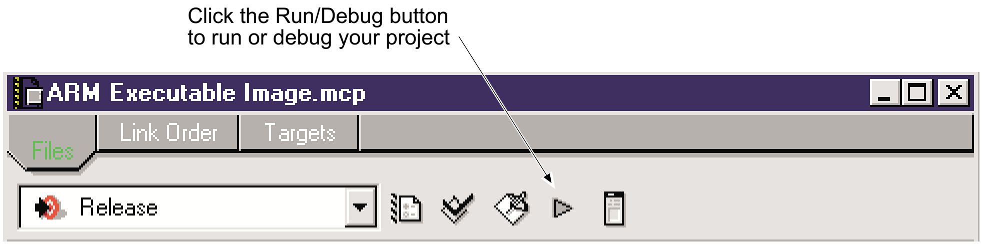


Figure 3-4 The Run/Debug button

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 ARM Runner target configuration panel (see *Configuring the ARM Runner* on page 9-153).

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.

3.3.2 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 Debugger* on page 9-136 for information on how to select an ARM debugger.

- *Configuring assembler and compiler language settings* on page 9-51 for information on how to enable debug table generation for the assembler and compilers.

——— **Note** ———

If your project is based on ARM-supplied stationery there are at least three separate build targets defined:

- Debug
- DebugRel
- 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.

If you are planning to release the same code you are debugging, select **Project** → **Set Current Target…** → **DebugRel**. This build target generates adequate debug information and provides good optimization.

———

3. Select **Enable Debugger** from the **Project** menu, if debugging is not already enabled.
4. 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 ARM Debugger configuration panel (see *Configuring the ARM Debugger* on page 9-136).

See the *ADS Debuggers Guide* for detailed information on using the ARM debuggers.

3.4 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* on page 3-12
- *Using the message window* on page 3-15.

3.4.1 Overview of the message window

The message window displays the following types of messages:

Errors Error messages are given by the compilers, 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 compilers, 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 postlinker. Notes are not displayed in the Errors & Warnings message window.

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 8-6 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.

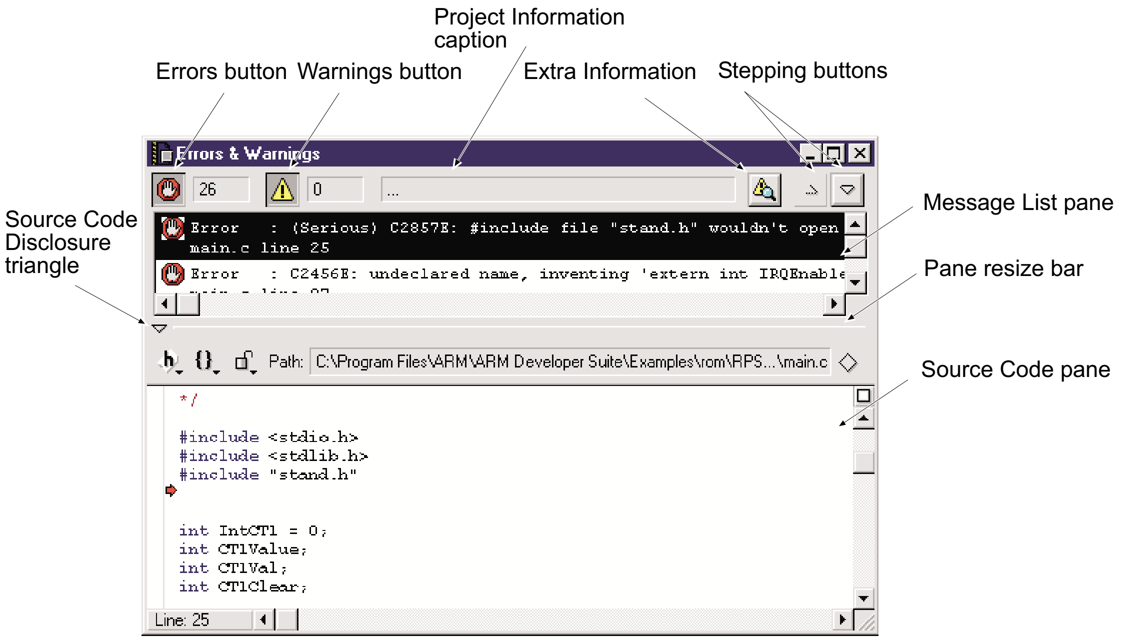
Some user interface items in the message window are not described here. See *Overview of the editor window* on page 5-3 for more information on:

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

Figure 3-5 on page 3-14 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.



```
Error   : (Serious) C2857E: #include file "stand.h" wouldn't open
main.c line 25
Error   : C2456E: undeclared name, inventing 'extern int IRQEnable
```

Path: C:\Program Files\ARM\ARM Developer Suite\Examples\rom\RPS...\main.c

```
*/

#include <stdio.h>
#include <stdlib.h>
#include "stand.h"


int IntCT1 = 0;
int CT1Value;
int CT1Val;
int CT1Clear;
```

Figure 3-5 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 3-16 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 3-16 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 3-16 for more information.

Message List pane

The Message List pane displays your messages. See *Viewing error and warning messages* on page 3-16 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 3-16 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.

3.4.2 Using the message window

The message window displays any error and warning messages given by the compilers, 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 3-16
- *Filtering error and warning messages* on page 3-16
- *Stepping through messages* on page 3-16
- *Correcting compilation errors and warnings* on page 3-17
- *Correcting link errors* on page 3-18
- *Searching library files* on page 3-19
- *Printing the message window* on page 3-19
- *Saving the message window* on page 3-20.

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 **Windows** 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 **Warning** button to deselect it and ensure that the **Error** button is selected.
- To view only warning messages in the Message List pane click the **Error** button to deselect it and ensure that the **Warning** button is selected.
- To view both error messages and warning messages in the Message List pane, ensure that both buttons are selected.

——— **Note** ———

Notes are not displayed in the Errors & Warnings message window.

———————————

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.

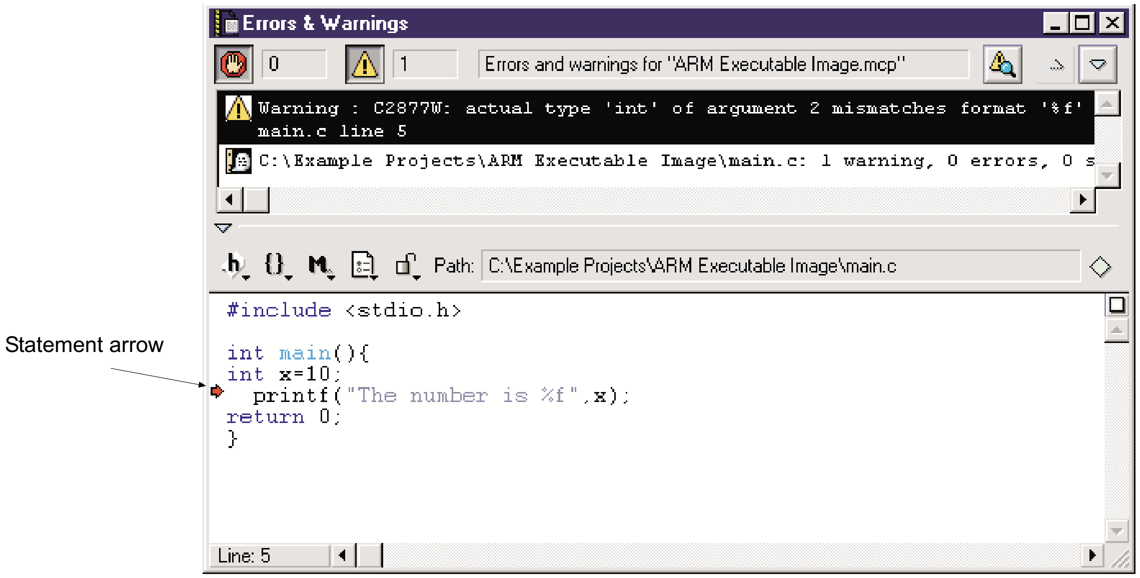
Correcting compilation errors and warnings

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 on page 3-14).
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 3-6 on page 3-18).

   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.



Errors and warnings for "ARM Executable Image.mcp"

```
Warning : C2877W: actual type 'int' of argument 2 mismatches format '%f'
main.c line 5
C:\Example Projects\ARM Executable Image\main.c: 1 warning, 0 errors, 0 s
```

Path: C:\Example Projects\ARM Executable Image\main.c

```
#include <stdio.h>

int main(){
int x=10;
  printf("The number is %f",x);
return 0;
}
```

Figure 3-6 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** pop-up menu, **Functions** pop-up menu, or the **Line Number** button to navigate the source code for a selected message. See *Overview of the editor window* on page 5-3 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 ATPCS options for the compilers and assembler. See *Configuring assembler ATPCS options* on page 9-58 and *Configuring compiler ATPCS options* on page 9-80 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 9-20 for more information.

  Check also that the setting for the **Use ARMLIB to find libraries** option is as you expect. If this option is selected, the ARM tools use the ARMLIB environment variable to search for libraries. See *Configuring linker options* on page 9-117 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 2-53 for more information.

- You have not correctly set up your linker output options. See *Configuring the ARM linker* on page 9-110 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 3-16 for information on how to move through the message window.

Searching library files

You can use armar to create a searchable text file of symbols in library files. For example, to create a text file of the symbol tables for all little-endian ARM C library files:

1. Open a DOS command prompt window.
2. Change directory to: `install_directory\lib\armlib`.
3. Type:
   ```
   armar -zs libname >> filename.txt
   ```
   for each of the little-endian libraries.
4. Edit the duplicate symbol names from the text file, if you want to.

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 4
Working with Files

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

4.1 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 operation 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 4-10)
- using the built-in file comparison features to compare and merge one or more files (see *Comparing and merging files and folders* on page 4-21).

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

- Chapter 2 *Working with Projects* for information on how source files fit into the project structure
- Chapter 5 *Editing Source Code* for information on how to use the CodeWarrior editor to edit files
- Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information on working with files that you have checked into a revision control system.

4.2 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 4-5
- *Opening files from the project window* on page 4-6
- *Opening header files from an editor window* on page 4-9.

——— **Note** ———

You cannot open libraries with the CodeWarrior editor.

———————

4.2.1 Creating a new file

You can create a new text file in the following ways:

- Using the **New Text file** menu command to create a new text file immediately. See *Using New Text File* on page 4-3.
- Using the **New…** menu command to create a new text file with the New dialog box. See *Using the New dialog* on page 4-4.

Using New Text File

To create a new source file:

1. Select **New Text File** from the **File** menu. A new, untitled editor window is displayed with the text insertion point on the first line of the window.
2. Enter your text or source code, as required. See Chapter 5 *Editing Source Code* for more information on editing text.
3. Save the text file. See *Saving files* on page 4-12 for more information.

——— **Note** ———

A new text file is not associated with any project. You must specifically add the new file to your project. See *Adding files to a project* on page 2-38 for more information.

———————

Using the New dialog

You can use the New dialog to create a new source file and optionally include it in a project. To create a new source 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 4-1).

   Note

   See *Creating a new project* on page 2-13 for more information on the **Project** tab. The **Object** tab is not used by the ARM version of the CodeWarrior IDE.

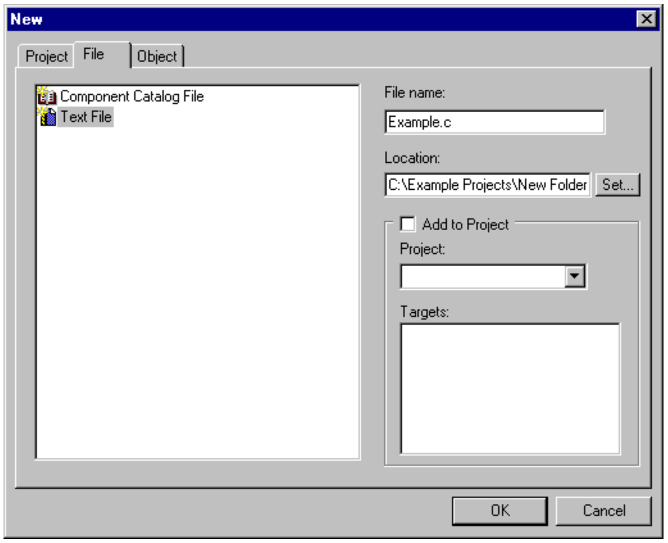


Figure 4-1 The New dialog box

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

   Note

   The Component Catalog File list item is not used by the ARM version of the CodeWarrior IDE.

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 *File Mappings panel* on page 9-42 for more information.

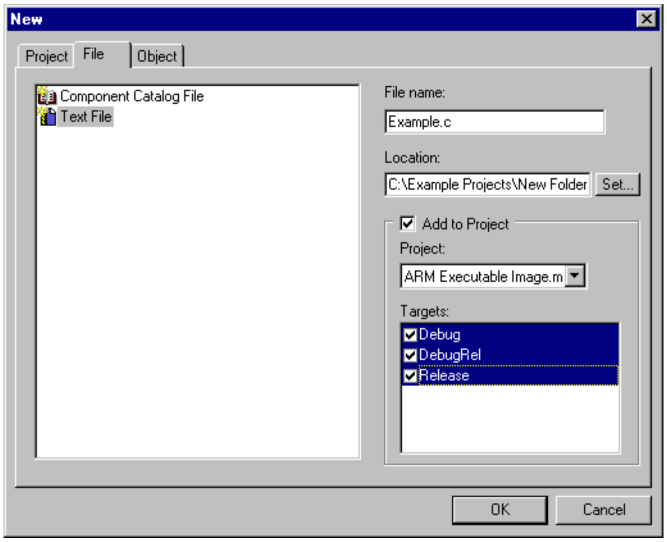
4. Enter a directory path to the new file in the Location field, or click **Set…** and select the directory from the standard file dialog.

5. Select the **Add to Project** checkbox if you want to add the new file to an existing project (Figure 4-2):

   a. Click the **Project** pop-up menu 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.

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.



File name: Example.c

Location: C:\Example Projects\New Folder

☑ Add to Project

Project: ARM Executable Image.m

Targets:
☑ Debug
☑ DebugRel
☑ Release

Figure 4-2 Add new file to project

4.2.2 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 2-13 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 4-3).

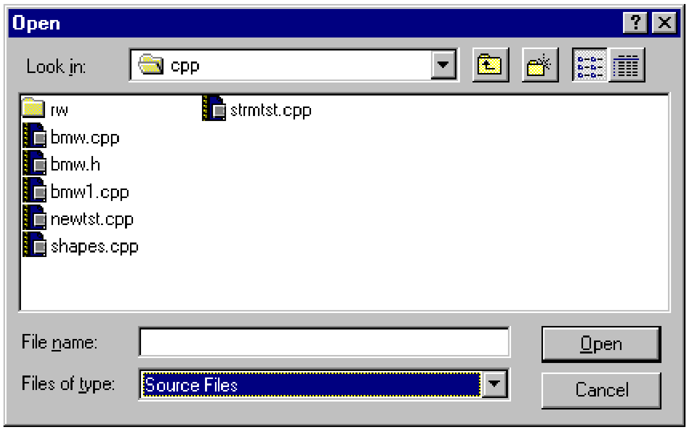


Figure 4-3 Open dialog box

2. Select **All Files** from the **Files of Type** pop-up menu. The list of displayed files changes to show all the files in the current directory, including text files.

3. Select the file you want to open and click **Open**. The CodeWarrior IDE opens the file in an editor window. See Chapter 5 *Editing Source Code* for more information on editing the file you have opened.

4.2.3 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 pop-up menu

Use this pop-up menu to open a text source file from within a collapsed group.

Using the Header Files pop-up menu

Use this pop-up 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 2-37 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 pop-up menu.

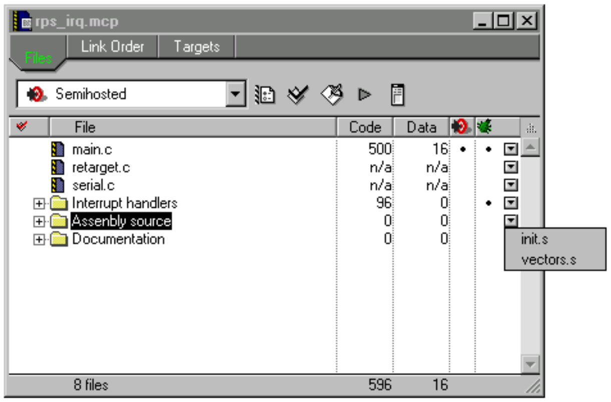
See *Overview of the project window* on page 2-4 for more information on the File column.

Opening files from the Group pop-up menu

You can open a source file from the pop-up 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** pop-up 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** pop-up menu is displayed that contains a menu item for each file within the group. Figure 4-4 shows an example.



File	Code	Data
main.c	500	16
retarget.c	n/a	n/a
serial.c	n/a	n/a
Interrupt handlers	96	0
Assenbly source	0	0
Documentation	0	0
8 files	596	16

Figure 4-4 Group pop-up menu

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

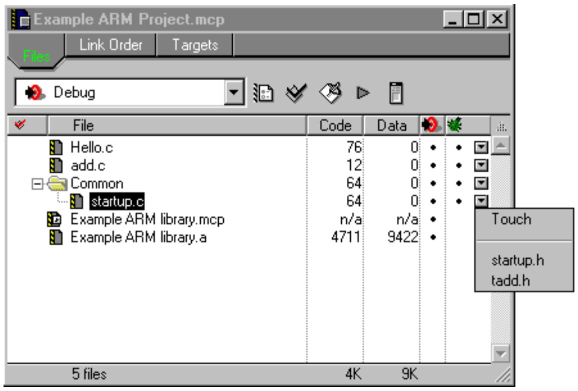
Opening header files from the Header Files pop-up menu

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 4-5 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 9-20 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 9-20 for more information on how the CodeWarrior IDE searches for user header files.



	File	Code	Data		
	Hello.c	76	0	•	•
	add.c	12	0	•	•
	Common	64	0	•	•
	startup.c	64	0	•	•
	Example ARM library.mcp	n/a	n/a	•	
	Example ARM library.a	4711	9422	•	
	5 files	4K	9K		

Figure 4-5 Header Files pop-up 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 4-10 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 2-47 for more information on touching files.

4.2.4 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 pop-up menu* on page 4-9
- *Opening a header file with the Find and Open menu item* on page 4-9
- *Switching between source and header files* on page 4-10.

Open a header file using the Header Files pop-up menu

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

1. Click the **Header Files** pop-up menu at the top left of the editor window (see Figure 5-10 on page 5-22). The pop-up menu lists all header files used by the source file.
2. Select a file from the list displayed in the **Header Files** pop-up menu to open it in a new editor window.

   Note

   If there are no files available in the pop-up menu, it means either:

   - the source file has not yet been compiled
   - 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. Select **Find and Open 'Filename'** from the **File** menu. 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. Select **Find and Open File** from the **File** menu. The CodeWarrior IDE displays a Find and Open File dialog (Figure 4-6)

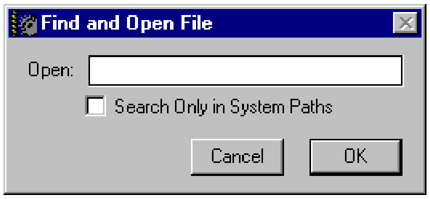


Figure 4-6 Find and open file dialog box

  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 9-20 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 9-20 for more information on configuring access paths.

4.3 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 4-14.

4.3.1 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.

4.3.2 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 4-13 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, press the keyboard shortcut Shift-Ctrl-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**
- **Run/Debug**.

You can use the **Save All Before Build** feature 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 All Before Build** option in *Configuring build settings* on page 8-6 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 Document As dialog box (Figure 4-7 on page 4-14).

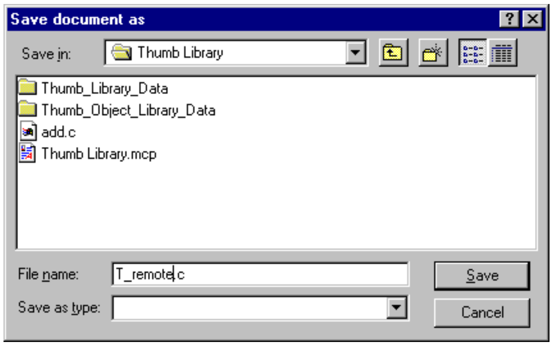


Figure 4-7 Save Document As dialog box

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* on page 4-14 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. When you open a text file originally created in a Mac OS, or UNIX text editor, the CodeWarrior IDE converts the text file internally to be compatible with Windows and corrects inconsistent line endings. When you finish editing the file, the CodeWarrior IDE saves the file in its original format.

See *Specifying Other Settings* on page 8-17 for more information on saving text files in a different text format.

4.3.3 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 document as dialog (Figure 4-8).

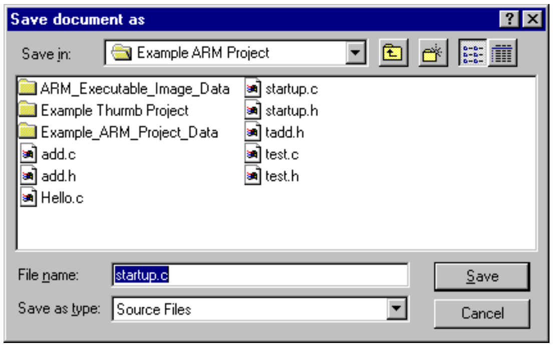


Figure 4-8 Save document as

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 a Save a Copy 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 4-28 for information on exporting a project to XML.

4.4 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*.

4.4.1 Closing project files

To close a project file, select **Close** from the **File** menu, or click the Windows close button. 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 2-18 for more information on saving project files.

4.4.2 Closing editor files

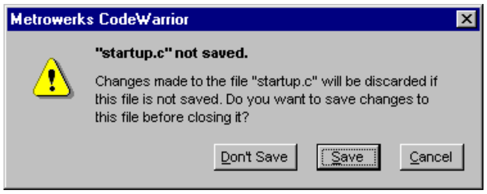
This section describes how to close editor files.

Closing one file

To close an editor window:

1. Select **Close** from the **File** menu, or click the close box for the 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 4-9).



"startup.c" not saved.

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

Figure 4-9 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 8-15 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.

Closing all files

To close all open editor windows:

1. Ensure that an editor window is the active window. You cannot close all files from the project window.
2. Select **Close All Editor Documents** from the **File** menu, or press Ctrl-Shift-W, or press Alt and click in the close box of an editor window. If any file contains unsaved changes, the editor prompts you to save information before closing the window that contains changes (see Figure 4-9 on page 4-16).

—— **Note** ——

Close All closes only the editor windows. The Find dialog box and project windows remain open.

4.5 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*.

4.5.1 Setting print options

To configure printing options:

1. Select **Print Setup…** 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 you have connected to your computer. See your printer and operating system documentation for more information on printer options.
3. Click **OK** to save your selected printer options.

4.5.2 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 two CodeWarrior-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

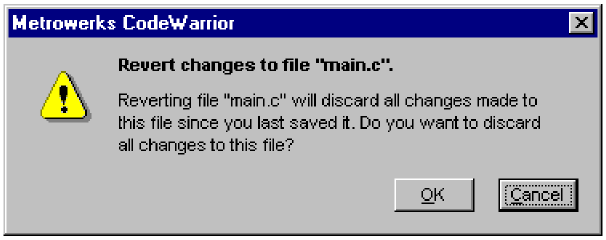
Select this option to print the editor window with syntax coloring. On a black and white printer, colors are printed as shades of gray. If this option is not selected, the CodeWarrior IDE prints the file in black and white without syntax coloring.

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

4.6 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 4-9).


Metrowerks CodeWarrior

Revert changes to file "main.c".

Reverting file "main.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 4-10 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 8-17 for more information.
- You can use the CodeWarrior IDE in conjunction with a version control system to recover previous checked-in versions of your editor files. See Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information.

4.7 Comparing and merging files and folders

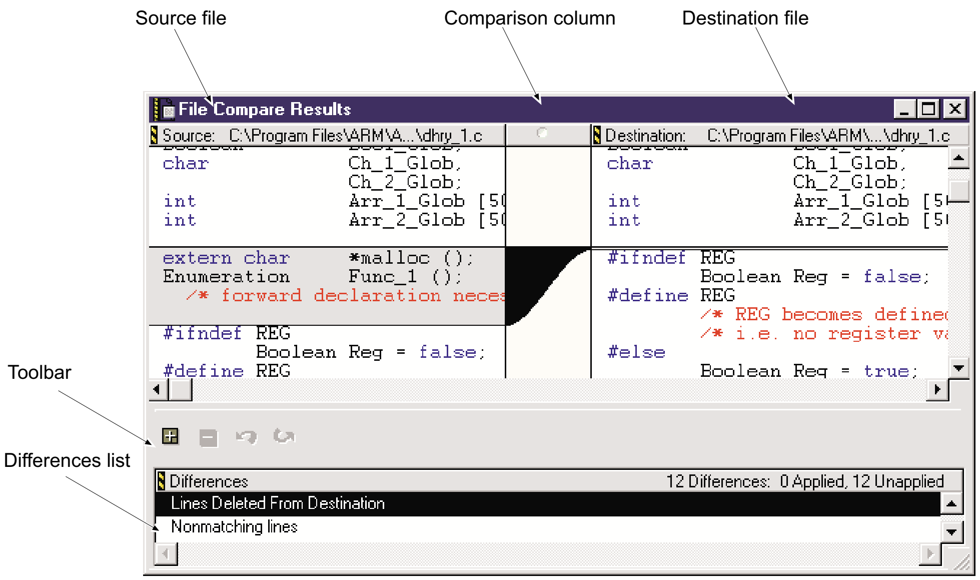
You can use the CodeWarrior IDE to compare two text files, mark the differences between the files, and apply changes between the files. In addition, you can compare the contents of two folders.

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

- *File comparison and merge overview* on page 4-21
- *Choosing files to compare* on page 4-23
- *Applying and unapplying differences* on page 4-24
- *Choosing folders to compare* on page 4-25
- *Comparing XML-formatted projects* on page 4-28.

4.7.1 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 4-11 shows an example.



Source: C:\Program Files\ARM\A...\dhry_1.c

```
char                Ch_1_Glob,
                    Ch_2_Glob;
int                 Arr_1_Glob [5(
int                 Arr_2_Glob [5(

extern char         *malloc ();
Enumeration         Func_1 ();
  /* forward declaration neces
#ifndef REG
        Boolean Reg = false;
#define REG
```

Destination: C:\Program Files\ARM\...\dhry_1.c

```
char                Ch_1_Glob,
                    Ch_2_Glob;
int                 Arr_1_Glob [5
int                 Arr_2_Glob [5

#ifndef REG
        Boolean Reg = false;
#define REG
        /* REG becomes definec
        /* i.e. no register va
#else
        Boolean Reg = true;
```

Differences 12 Differences: 0 Applied, 12 Unapplied

Lines Deleted From Destination

Nonmatching lines

Figure 4-11 The Compare Files 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 8-37 for information on how to customize the toolbar.

Table 4-1 shows the control icons.

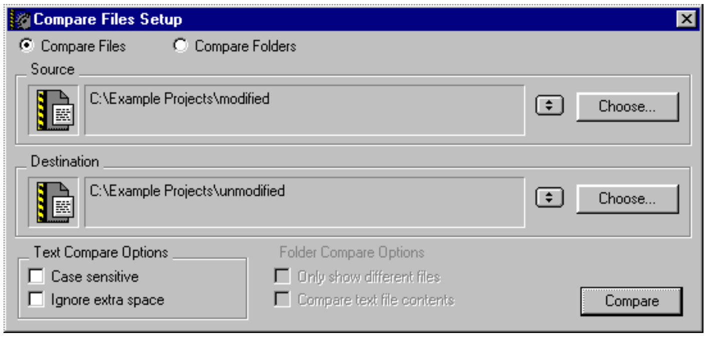
Table 4-1 Toolbar control icons

Control	Action
⊞	Apply difference
⊟	Unapply difference
◂Z	Undo
Z▸	Redo

4.7.2 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 4-12).



Compare Files Setup

[✓ Compare Files] Compare Folders

Source

C:\Example Projects\modified

Destination

C:\Example Projects\unmodified

Text Compare Options
- ☐ Case sensitive
- ☐ Ignore extra space

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

Figure 4-12 Compare Files Setup window

2. Ensure that the **Compare Files** radio button is selected. This enables you to select files, rather than folders, by using the **Choose** buttons.

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

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

4. 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.

   The folder compare options are not available when the **Compare Files** radio button is selected. See *Choosing folders to compare* on page 4-25 for information on the folder comparison options.

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

Comparing open editor windows

To compare two files that are already open in editor windows:

1. Select **Compare Files** from the **Search** menu. The CodeWarrior IDE displays a Compare Files Setup dialog that prompts you for a source file and a destination file to compare (Figure 4-12 on page 4-23).

2. Click the **Editor Files** pop-up menu next to the source and destination paths (Figure 4-12 on page 4-23). The CodeWarrior IDE displays a list of all open editor windows.

3. Select a file name from the pop-up menu to make it the source or destination file.

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

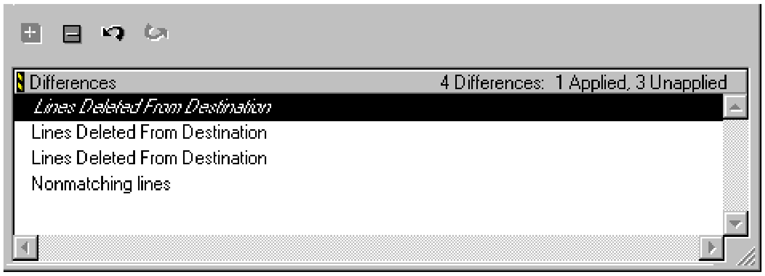
4.7.3 Applying and unapplying differences

Use the Comparison window toolbar and Differences list 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** 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.



Differences 4 Differences: 1 Applied, 3 Unapplied

Lines Deleted From Destination
Lines Deleted From Destination
Lines Deleted From Destination
Nonmatching lines

Figure 4-13 Applied difference

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.

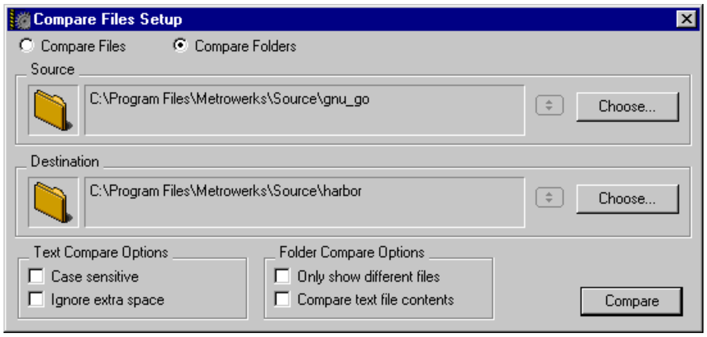
——— **Note** ———

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.

4.7.4 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 4-14).



Source: C:\Program Files\Metrowerks\Source\gnu_go

Destination: C:\Program Files\Metrowerks\Source\harbor

Text Compare Options
- [ ] Case sensitive
- [ ] Ignore extra space

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

Figure 4-14 Compare Folders Setup dialog box

2. Ensure that the **Compare Folders** radio button is selected. This enables you to select folders, rather than files, by using the **Choose** buttons, and enables the Folder Comparison Options described below.

3. 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.

4. 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 4-15 on page 4-27). 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 4-23 for information on the file comparison options.

5. Click **Compare** to compare the two folders. The CodeWarrior IDE displays the Folder Compare Results window (Figure 4-15 on page 4-27). 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.

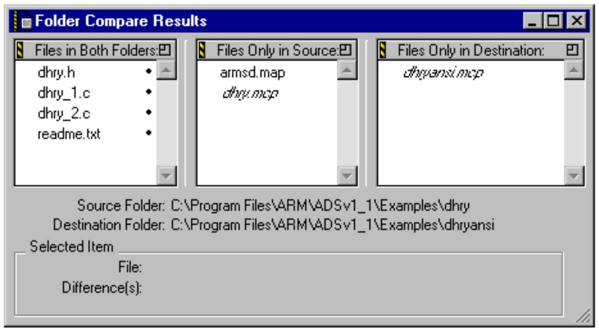


Figure 4-15 Folder Compare Results window

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.

6. 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.

7. 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

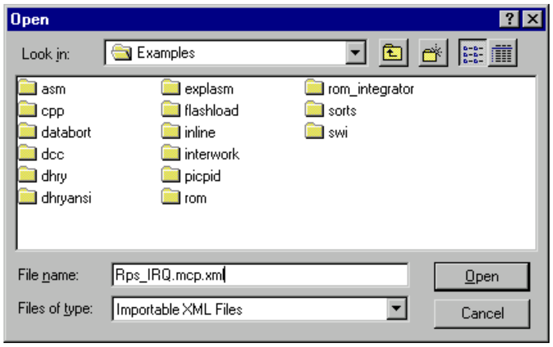
You can click on a zoom box ( ) for any of the three lists to expand them to fill the window. Click the zoom box again to collapse the lists back to their original size.

4.7.5 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 project.

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

1. Convert the projects you want to compare 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 4-16).



File name: Rps_IRQ.mcp.xml

Files of type: Importable XML Files

Figure 4-16 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 4-12 on page 4-23).

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 4-23 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 4-11 on page 4-21 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 4-24 for more information.

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

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

8. Select the XML file you want to import and click **Open**. The CodeWarrior IDE displays a Save As dialog 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.

Chapter 5
Editing Source Code

This chapter describes how to use the CodeWarrior IDE text editor to edit your source code. It contains the following sections:

5.1 About editing source code

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

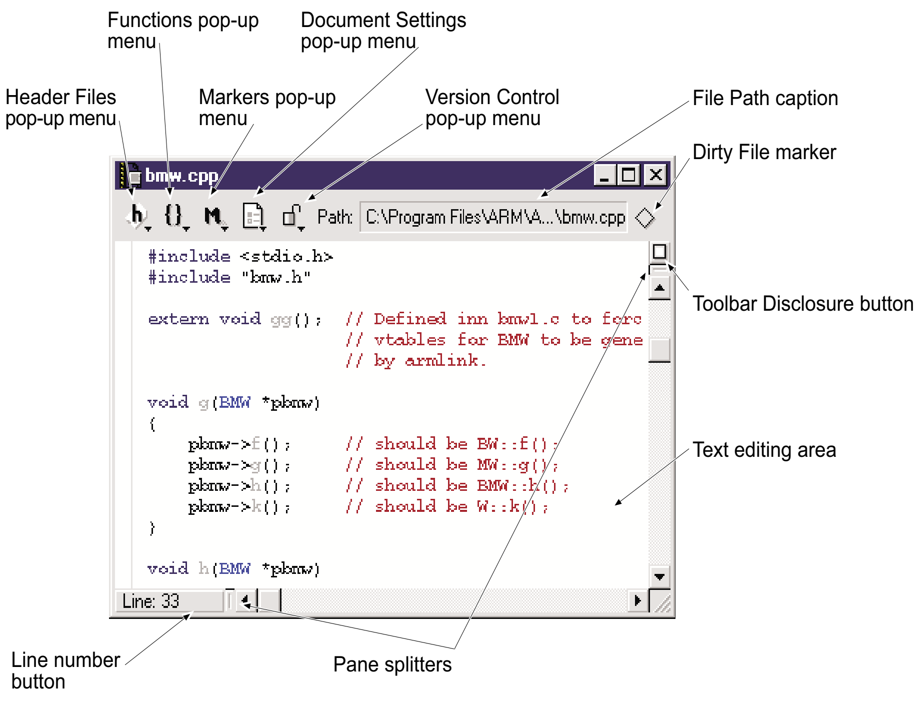
- pop-up menus on every editor window for opening header files and quickly navigating among functions
- integrated version control menus that enable you to work with your version control system from within the CodeWarrior IDE
- syntax highlighting that formats source code for easy identification of comments 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 8-15 for more information. See also:

- Chapter 4 *Working with Files* for information on basic file operations such as opening, saving, and comparing source files
- Chapter 6 *Searching and Replacing Text* for information on searching and replacing text in source files
- Chapter 7 *Browsing Source Code* for information on using the CodeWarrior source code browser.

5.2 Overview of the editor window

The editor window provides pop-up menus and other controls that enable you to perform basic editing operations. Figure 5-1 shows an example of the CodeWarrior editor window.



```
#include <stdio.h>
#include "bmw.h"

extern void gg();   // Defined inn bmwl.c to forc
                    // vtables for BMW to be gene
                    // by armlink.

void g(BMW *pbmw)
{
    pbmw->f();      // should be BW::f();
    pbmw->g();      // should be MW::g();
    pbmw->h();      // should be BMW::h();
    pbmw->k();      // should be W::k();
}

void h(BMW *pbmw)
```

Figure 5-1 The editor window

The major interface components of the editor window are:

Header Files pop-up menu

You can use the **Header Files** pop-up menu, shown in Figure 5-10 on page 5-22, to either:

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

See *Touching and untouching files* on page 2-47 for more information.

Functions pop-up menu

You can use the **Functions** pop-up menu shown in Figure 5-5 on page 5-18 to jump to a specific function in another text file within your source code. See *Using the Functions pop-up menu* on page 5-17 for more information.

Markers pop-up menu

You can use the **Markers** pop-up menu (Figure 5-2) 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.

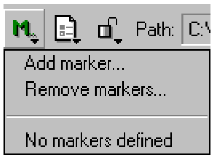


Figure 5-2 The Marker pop-up menu

See *Using markers* on page 5-19 for more information.

Document Settings pop-up menu

You can use the **Document Settings** pop-up menu (Figure 5-3), to turn color syntax highlighting on or off for the current file, and to set the method for saving the file.

See *Syntax Coloring* on page 8-22 for details of how to modify syntax coloring.

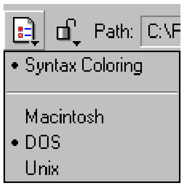


Figure 5-3 The Document Settings pop-up menu

Version Control pop-up menu

The **Version Control** pop-up menu indicates the read/write revision control database status of the current file. You can modify the file if the pop-up icon box shows the **Unlocked** icon or the **Read/Write** icon. The icons and their meanings are described in *Performing common VCS operations* on page 10-7.

Depending on the VCS system your are using, you can use this pop-up menu to:

- get a new copy of your file
- checkout the file for modification
- make the file writable so you can make changes without performing a checkout.

See *Performing common VCS operations* on page 10-7 for more information about revision control system software.

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 5-1 on page 5-3.

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 5-4 on page 5-8 shows an editor window with multiple panes.

See *Splitting the window into panes* on page 5-8 for more information on pane splitter controls.

Line Number button

The Line Number box shown in Figure 5-1 on page 5-3 displays the number of the line that contains the text insertion point. You can also use this button to go to another line in the file.

See *Going to a specific line* on page 5-20 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 bottom of the editor window.

See *Displaying window controls* on page 5-7 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 5-10.

5.3 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 5-8
- *Saving editor window settings* on page 5-9.

See *Customizing toolbars* on page 8-37 for more information on configuring the editor window toolbar.

5.3.1 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 8-19 for more information.

5.3.2 Displaying window controls

The toolbar comprises the row of pop-up menus and controls that appears along the top of the editor window. Use the **Toolbar Disclosure** button, shown in Figure 5-1 on page 5-3, 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 pop-up menu controls along the bottom of the editor window.

——— **Note** ———

- If you have customized the editor window toolbar, your custom toolbar items are not displayed at the bottom 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.

———————

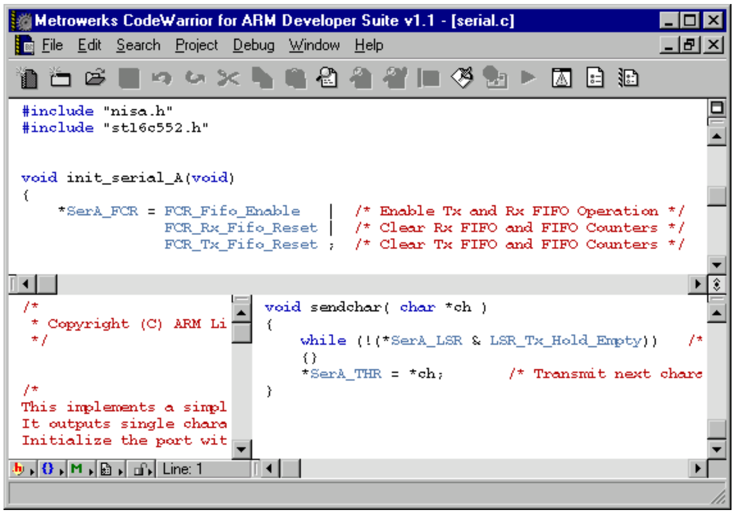
To re-show the toolbar if it is hidden, click the **Toolbar Disclosure** button. The toolbar is displayed along the top of the editor window.

See *Customizing toolbars* on page 8-37 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 8-38 for more information.

5.3.3 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 5-4 shows an example. The following sections describe how to create, resize, and remove multiple panes.


Metrowerks CodeWarrior for ARM Developer Suite v1.1 - [serial.c]
File Edit Search Project Debug Window Help
Line: 1

```
#include "nisa.h"
#include "st16c552.h"


void init_serial_A(void)
{
    *SerA_FCR = FCR_Fifo_Enable   |  /* Enable Tx and Rx FIFO Operation */
                FCR_Rx_Fifo_Reset |  /* Clear Rx FIFO and FIFO Counters */
                FCR_Tx_Fifo_Reset ;  /* Clear Tx FIFO and FIFO Counters */
```

```
/*
 * Copyright (C) ARM Li
 */


/*
This implements a simpl
It outputs single chara
Initialize the port wit
```

```
void sendchar( char *ch )
{
    while (!(*SerA_LSR & LSR_Tx_Hold_Empty))   /*
    {}
    *SerA_THR = *ch;        /* Transmit next charc
}
```

Figure 5-4 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.

5.3.4 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.

5.4 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 5-11
- *Selecting text* on page 5-12
- *Moving text with drag-and-drop* on page 5-13
- *Balancing punctuation* on page 5-14
- *Shifting text left and right* on page 5-15
- *Undoing changes* on page 5-15
- *Controlling color* on page 5-16.

5.4.1 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 5-1 shows the keystrokes you can use to move the insertion point in a file.

Table 5-1 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 5-2 shows the keystrokes you can use to scroll to different locations in a file, without moving the insertion point.

Table 5-2 Scroll with the keyboard

To scroll to…	Keystroke
Previous page	Page Up
Next page	Page Down
Beginning of file	Ctrl-Home
End of file	Ctrl-End
Previous line	Ctrl-up arrow
Next line	Ctrl-down arrow

5.4.2 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 5-10 for ways to move the insertion point in an editor window.

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*, 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 B-5 for more information on these commands.

5.4.3 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 5-3 shows the keystrokes for selecting text, starting at the current insertion point.

Table 5-3 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 5-14 for more information.

Selecting text using the mouse

Table 5-4 gives a summary of how to select text with the mouse.

Table 5-4 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** pop-up 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.

5.4.4 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 would like 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 8-15 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 4-3 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.

5.4.5 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.

Using automatic balancing

You can have the editor check for balanced punctuation automatically. See *Other settings* on page 8-9 for more information on how to configure the CodeWarrior IDE to check punctuation automatically as you type.

5.4.6 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 5-12).
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 8-19 for information on configuring the number of spaces defined for a tab character.

5.4.7 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 8-9 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 4-20 for more information.

5.4.8 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 *Syntax Coloring* on page 8-22 for information on configuring color syntax options.

5.5 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 5-18
- *Using markers* on page 5-19
- *Going to a specific line* on page 5-20
- *Using Go Back and Go Forward* on page 5-21.
- *Opening a related header file* on page 5-21.

See also Chapter 7 *Browsing Source Code* 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 *Modifying key bindings* on page 8-34 for more information on how to the change key bindings that move the text insertion point around in a file.

5.5.1 Finding a function

You can use the **Functions** pop-up menu to find a specific function within the source file currently displayed in the editor window.

Note

- If the pop-up menu is empty, the current editor file is not a source code file.
- You cannot use the **Functions** pop-up menu to navigate through ARM assembly language code.

Using the Functions pop-up 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** pop-up menu (Figure 5-5). The pop-up 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** pop-up menu.

Note

You can change the default order of the functions in the **Functions** menu with the **Sort Function Popup** configuration option. See *Editor settings* on page 8-15 for more information.

If the function name in the pop-up list has a bullet next to it, it means that the text insertion point is currently located within the definition for that function.

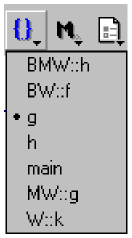


Figure 5-5 The Functions pop-up menu

5.5.2 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 7-27). Otherwise, the CodeWarrior IDE displays a message window (see *Using the message window* on page 3-15).

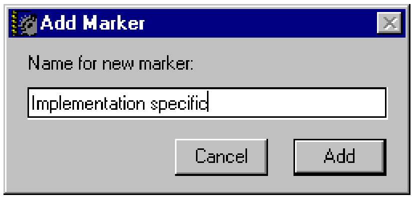
5.5.3 Using markers

You can add or remove a marker in any of your text files using the facilities built into the CodeWarrior editor. Markers function as bookmarks for setting places in your file that you can jump to quickly, or for leaving notes to yourself about work in progress on your code.

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** pop-up menu. The CodeWarrior IDE displays an Add Marker dialog box (Figure 5-6).



Implementation specific

Figure 5-6 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** pop-up menu (Figure 5-7).

Note

If you select some text in a source file, then select **Add Marker**, the selected text will appear as the new marker name in the **Add Marker** dialog. This is useful for quickly adding specific functions or lines as markers.

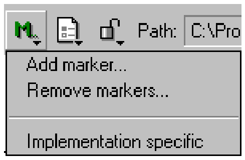
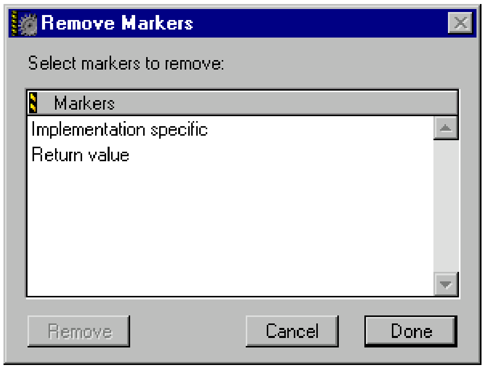


Figure 5-7 Example text file with a marker added

Removing a marker

To remove a marker:

1. Click the **Markers** pop-up menu and select the **Remove markers** command. CodeWarrior displays the Remove Markers dialog box (Figure 5-8).



Select markers to remove:

Markers
Implementation specific
Return value

Figure 5-8 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** pop-up 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.

5.5.4 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 5-9 on page 5-21).

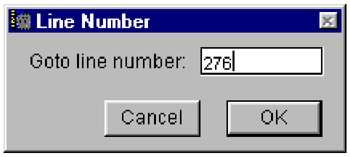


Figure 5-9 Line Number dialog box

2. Enter the number of the line you want to jump to.

3. Click **OK**.

5.5.5 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 7-22 for details on using these commands. See Chapter 7 *Browsing Source Code* for more information on using the browser.

5.5.6 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** pop-up menu
- using a keyboard shortcut.

Using the Header Files pop-up menu

You can use the **Header Files** pop-up 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 2-47 for more details.

To use the **Header Files** pop-up 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** pop-up menu icon to display the menu (Figure 5-10).

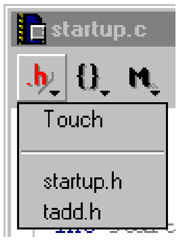


Figure 5-10 The Header Files pop-up menu

Note

- You can also open the Header Files pop-up 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 4-3 for more information.

Chapter 6
Searching and Replacing Text

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

6.1 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 6-1 on page 6-4, 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 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.

6.2 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 dialog* on page 6-4
- *Using batch searches* on page 6-6.

6.2.1 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 toward the beginning of the file for the previous occurrence of the text string.

Finding text in another window

This method is useful if you are working with a file in an editor window and you want to find occurrences of a text string in another file. To find text in another editor window:

1. Select an instance of the text you want to find.
2. Select **Enter 'Find' String** from the **Search** menu. The editor enters the selected text into the Find text field of the Find dialog box.

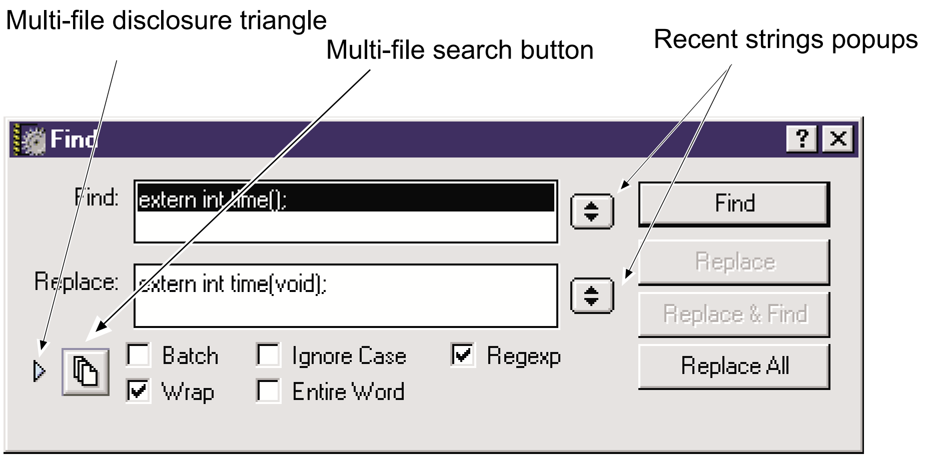
3. Click on the editor window that you want to search to make it active.

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.

6.2.2 Finding and replacing text with the Find dialog

To find and replace text with the Find dialog box:

1. Select **Find…** from the **Search** menu. The Find dialog box is displayed (Figure 6-1).



Find: extern int time();
Replace: extern int time(void);
Batch: off
Wrap: on
Ignore Case: off
Entire Word: off
Regexp: on

Figure 6-1 The Find dialog box search and replace section

2. Ensure that multi-file searching is turned off so that only the active editor window is searched. Multi-file searching is controlled by:

   Multi-File Search Disclosure triangle

   Click this triangle to toggle the display of the Multi-File Search section of the Find dialog box.

   Multi-File Search button

   Click this button to enable or disable the options in the Multi-File Search section of the Find dialog box. When the **Multi-File Search** button is not depressed, the items in the Multi-File Search section of the Find dialog box are disabled.

See *Finding and replacing text in multiple files* on page 6-8 for more information on multi-file searching using the Find dialog box.

3. Type a text string into the Find text field, or select a string from the **Recent Strings** pop-up menu.

   The **Recent Strings** pop-up menu contains strings that have previously been used for searches. Select an item in one of these pop-up menus to place it into the corresponding text box.

   You can use Cut, Copy, and Paste commands to edit text in the Find text field.

   Note

   See *Searching for special characters* on page 6-6 for information on how to search for a Return or Tab character.

4. Type a text string into the Replace text field, or select a string from the **Recent Strings** pop-up menu, if you want to replace the found string.

5. Select the search options you want:

   Batch Select this option to display the results of the search in a Search Results message window (see Figure 6-3 on page 6-7). See *Using batch searches* on page 6-6 for more information on using the **Batch** option when searching.

   Ignore Case

   Select this option to treat uppercase and lowercase text in the search string as identical. When this option is not selected, uppercase and lowercase text are distinct.

   Regexp 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 6-16 for more information.

   Wrap Select this option if you want the search to wrap the end of the file. CodeWarrior searches from the current insertion point to the end of the file, and continues to the insertion point.

   If you search multiple files with this option enabled, the CodeWarrior IDE searches from the first file in the file list after it reaches the last file.

   Entire 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.

6. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace operation:
   - Click **Find**, or select **Find…** or **Find Next** from the **Search** menu to search forward from the text insertion point in the active editor window.
   - Select **Find Previous** from the **Search** menu, or press the Shift key and click **Find**, to search backwards from the text insertion point in the file.
   - Click **Replace** to replace found text with the text in the Replace text field.
   - Click **Replace & Find** to replace found text and find the next occurrence of the find string. Press the Shift key to replace and search backwards.
   - Click **Replace All** 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 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 **Regexp** option, and enter `\t` for Tab or `\r` for Return into the field.

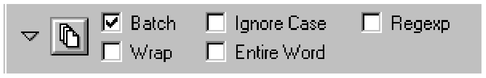
Using regular expressions alters the way in which the CodeWarrior IDE locates a string match. See *Using grep-style regular expressions* on page 6-15 for more details.

6.2.3 Using batch searches

The batch search option in the Find dialog box enables you to collect all successful matches of your search text in one window for easy reference.

To use batch searches:

1. Select **Find…** from the **Search** menu.
2. Ensure that the **Batch** option is selected (Figure 6-2).

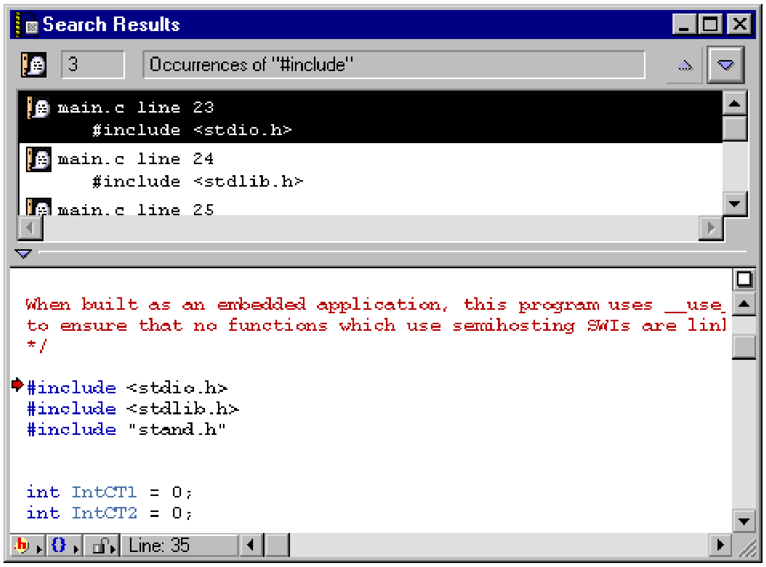


[✓] Batch [ ] Ignore Case [ ] Regexp
[ ] Wrap [ ] Entire Word

Figure 6-2 Batch checkbox

3. Enter your search criteria and options. See *Finding and replacing text with the Find dialog* on page 6-4 for details.

4. Click **Find**. The search results are displayed in a Search Results message window (Figure 6-3 on page 6-7).



```
main.c line 23
    #include <stdio.h>
main.c line 24
    #include <stdlib.h>
main.c line 25
```

```
When built as an embedded application, this program uses __use_
to ensure that no functions which use semihosting SWIs are lin]
*/

#include <stdio.h>
#include <stdlib.h>
#include "stand.h"


int IntCT1 = 0;
int IntCT2 = 0;
```

Figure 6-3 Batch search results

5. 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 an editor window. See *Using the message window* on page 3-12 for more information on the features of the message window.

6.3 Finding and replacing text in multiple files

The CodeWarrior IDE Find dialog enables you to search multiple files for the occurrence of text strings. The Multi-File Search section of the Find dialog box enables you to specify searches through:

- source files in a project
- header files in the project
- any additional files you specify.

In addition, you can use the Multi-File Search section of the Find dialog box to save sets of search files for future use.

This section describes:

- *Using multi-file search*
- *Using file sets* on page 6-12.

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 7-22 for more information.

6.3.1 Using multi-file search

This section describes how to perform multi-file searches. See *Using file sets* on page 6-12 for more information on configuring multi-file searches. To search for text in multiple files:

1. Select **Find…** from the **Search** menu. The Find dialog box is displayed (see Figure 6-1 on page 6-4).
2. Click the **Multi-File Search Disclosure** triangle to the left of the **Multi-File Search** button so that the triangle points down. The CodeWarrior IDE displays the Multi-File Search section of the Find dialog box (Figure 6-4 on page 6-9).

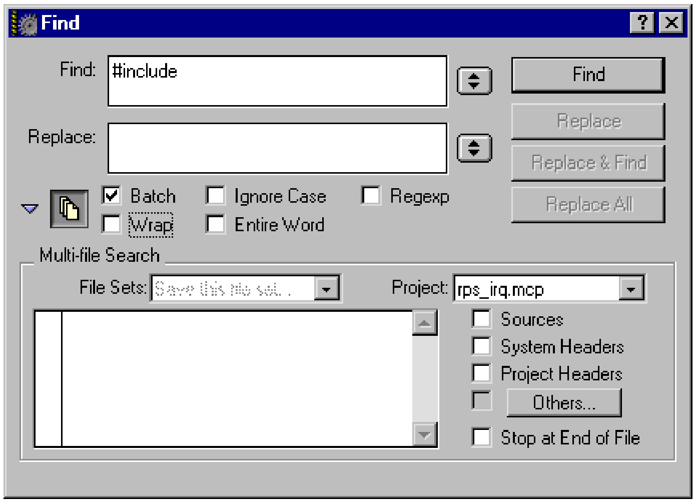


Figure 6-4 The Find dialog box with multi-file search options

3. Click the **Multi-File Search** button to enable multi-file searching.
   - When the **Multi-File Search** button is on, the button appears to be depressed.
   - When the **Multi-File Search** button is off, the button appears to be raised.
4. Select the search options you want. You can specify the same search options as for a single file search. See *Finding and replacing text with the Find dialog* on page 6-4 for details.
5. Click the **Project** pop-up menu (Figure 6-5) to select the project file with which you want to perform your search. The **Project** pop-up menu displays a list of all currently open projects.

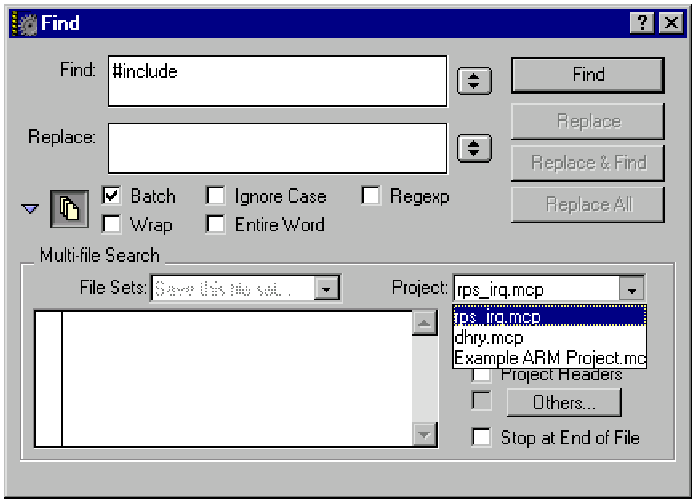


Figure 6-5 Project pop-up menu

6. Select from the following options to specify the files you want to search. The file types you select are added to the search file list. See *Using file sets* on page 6-12 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.

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 9-20 for more information.

Note

- You must have successfully compiled your source files in order to search the system header 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 2-77 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** ————

- You must have successfully compiled your source files in order to search the project header 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 2-77 for more information.

————————

Others…

Click the **Others…** button and use the standard file dialog box to add non-project files to your search file list.

——— **Note** ————

The standard file dialog does not list files with filename extensions that are not recognized by the CodeWarrior IDE. To view all available files, enter `*.*` in the Filename text field of the dialog box. Alternatively you can drag and drop text files with any file name extension directly onto the **Others…** button.

————————

Stop at End of File

Select this option to search each file in the file list individually. When the CodeWarrior IDE reaches the end of a file, it stops searching and beeps. Select **Find in Next File** from the **Search** menu to continue the search.

Deselect this option to treat all files in the file list as one large file. When the CodeWarrior IDE reaches the end of a file, it searches the next file in the file list until the search text is found. Select **Find…**, **Find Next**, or **Find Previous** to resume searching. The CodeWarrior IDE beeps when it reaches the end of the last file to search.

7. Use the dialog box buttons, or menu items from the **Search** menu, to start the find, or find and replace operation. See *Finding and replacing text with the Find dialog* on page 6-4 for details.

6.3.2 Batch searching through text files

By default, the CodeWarrior multi-files search dialog does not display text files with arbitrary filename extensions, such as `.txt`. To search through text files you can either:

- Specify `*.*` as the filename in the Select files to search… dialog.
- Drag and drop the files you want to search directly onto the **Others…** button from either the Windows interface, or from Windows Explorer (see *The Find dialog box with multi-file search options* on page 6-9). From a project window you can right-click on a file and select Open in Windows Explorer to open Windows explorer.

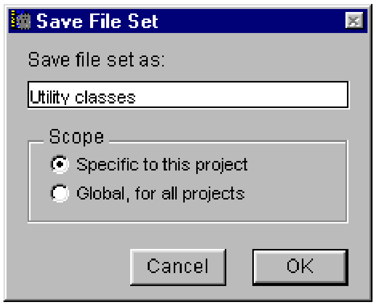
6.3.3 Using file sets

You can use the multi-file search section of the Find dialog box to save sets of frequently searched files for later use.

Saving a file set

To save a file set for use in future multi-file searches:

1. Ensure that the multi-file search section of the Find dialog is displayed, and select the files you want to search. See *Using multi-file search* on page 6-8 for details.
2. Select **Save this File Set** from the **File Sets** pop-up menu (see Figure 6-4 on page 6-9). The CodeWarrior IDE displays the Save File Set dialog box (on page 6-12).



Utility classes

[● Specific to this project] Global, for all projects

Figure 6-6 The Save File Set dialog box

3. Enter a name for the file set in the Save file set as text field, and select the scope of the file set:

   Specific to this project

   Select this option if you plan to use this file set only with the current project. The CodeWarrior IDE stores the file set in the project.

   Global, for all projects

   Select this option if you want to use this file set with other projects. The CodeWarrior IDE stores the file set in its preferences file, enabling all projects to use it.

4. Click **OK** to save the file set.

Choosing a file set

To select a previously-saved file set to include in your search, click on the **File Sets** pop-up menu and select a file set from the menu. The files in the file set are added to the search file list.

Removing a file set

To remove a file set from the list of saved search file sets:

1. Ensure that the multi-file search section of the Find dialog is displayed.

2. Select **Remove a file set** from the **File Sets pop-up** menu (see Figure 6-4 on page 6-9) to remove a previously-saved file set. The CodeWarrior IDE displays the Remove File Sets dialog box (Figure 6-7 on page 6-13).

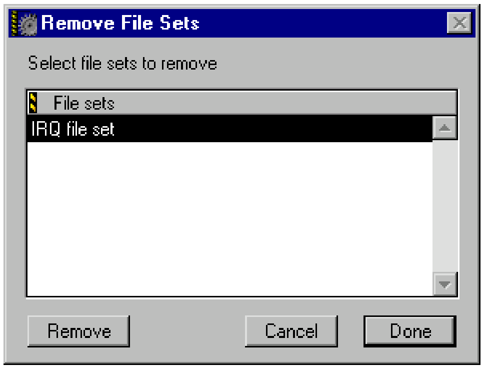


Figure 6-7 Remove File Sets dialog box

3. Select the file set you want to remove and click **Remove**. The CodeWarrior IDE removes the file set.

4. Click **Done** to return to the Find dialog box.

6.4 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 6-16.

6.4.1 Special operators

Table 6-1 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 6-16 for more information.

Table 6-1 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 6-1 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.

6.4.2 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 **Regexp** option is selected (Figure 6-8).

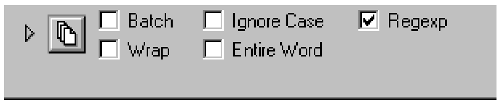


Figure 6-8 Regexp checkbox

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 6-1 on page 6-15. 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 6-2 shows some simple examples.

Table 6-2 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* on page 6-18 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 7
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 ARM C and C++ compilers and the ARM assembler 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:

7.1 About working with the browser

The CodeWarrior browser enables you to view symbolic information, generated by the ARM compilers and assembler, on the objects defined in your code, and the relationships between those objects.

Browser windows provide three *views* on the objects in the current build target, and enables 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 ARM compilers produce 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.

7.1.1 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 7-8 on page 7-17 shows an example of a contents view.

See *Viewing data by type with the Contents view* on page 7-17 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 7-3 on page 7-9 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 7-8 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 7-9 on page 7-19 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 7-18 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 Contextual menu* on page 7-22 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 7-1 summarizes the general viewing choices available when using the browser.

Table 7-1 Browser viewing options

Viewing style	Wide focus	Narrow focus
Comprehensive	Contents	Not applicable
Inheritance-based	Multi-class hierarchy	Single-class hierarchy

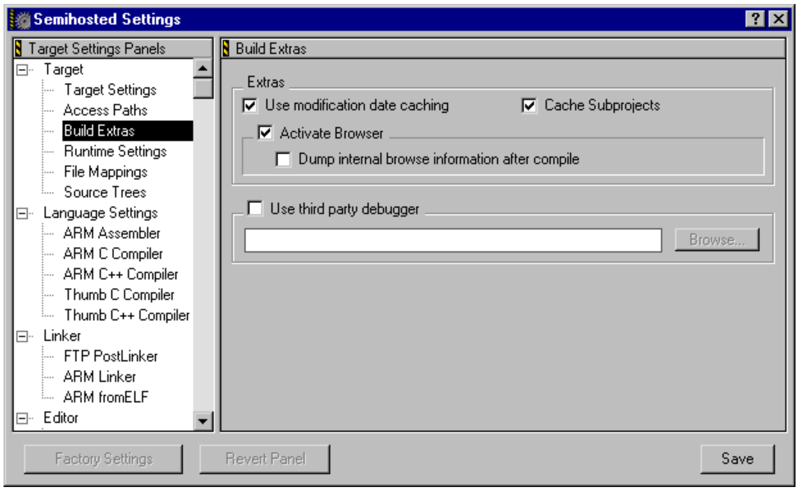
The browser-related menu commands in the **Windows** menu (**Browser Contents window**, **Class Hierarchy window**, and **New Class Browser**) display wide-focus views. After you have selected a wide view, you can use a context-sensitive menu to focus on a particular class.

7.2 Activating the browser

You must activate the browser and recompile your project before browser information is available. To activate the browser:

1. Ensure that your project window is the active window and click the **Target Settings** icon in the toolbar. The CodeWarrior IDE displays the Target settings panel for your project.

2. Click **Build Extras** in the Target Settings Panels list. The CodeWarrior IDE displays the Build Extras panel (Figure 7-1).



Extras
- [✓] Use modification date caching
- [✓] Cache Subprojects
- [✓] Activate Browser
  - [ ] Dump internal browse information after compile
- [ ] Use third party debugger

Figure 7-1 Target settings panel

3. Select **Activate Browser** and click **Save**. The CodeWarrior IDE marks your source files for recompilation.

4. Close the Target Settings panel and select **Make** or **Bring up to Date** from the **Project** menu, or click the **Make** button in your project toolbar to rebuild the project.

   When the project is rebuilt, the ARM compilers generate a database of information about your code, and about the relationships between various parts of your code, such as inheritance hierarchies.

   ——— **Note** ———

   You can also selectively compile individual source files to generate browser information for those files only.

See *Configuring browser options* on page 7-6 for more details on browser settings and options

7.2.1 Configuring browser options

Browser-related menu items and browser-specific options are available only when you activate the browser. See *Activating the browser* on page 7-5 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 **Windows** 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 *Browser Display* on page 8-14 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** ———

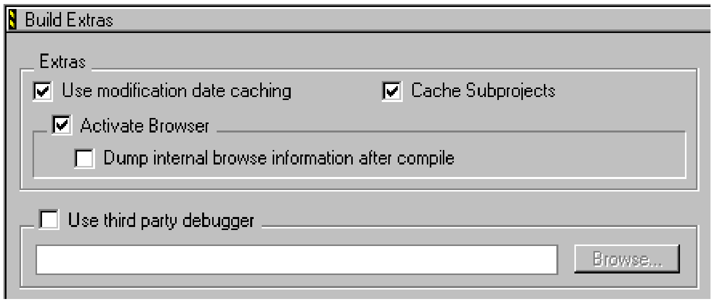
This option is selected by default.

————————

To enable subproject caching:

1. Display the Target Settings panel for the project you want to configure (see *Displaying Target Settings panels* on page 9-8) for more information.

2. Click **Build Extras** in the Target Settings Panels list and click the **Target** tab to display the configuration panel (Figure 7-2).



Build Extras

Extras

- [✓] Use modification date caching
- [✓] Cache Subprojects
- [✓] Activate Browser
  - [ ] Dump internal browse information after compile
- [ ] Use third party debugger

Figure 7-2 Build Extras settings 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 9-36 for more information on the Build Extras panel.

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

7.3 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 7-17
- *Viewing class hierarchies and inheritance with the hierarchy view* on page 7-18.

7.3.1 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 or Thumb C++ compiler to process all source and header files containing classes. By default, header files are mapped to the ARM and Thumb C compilers.

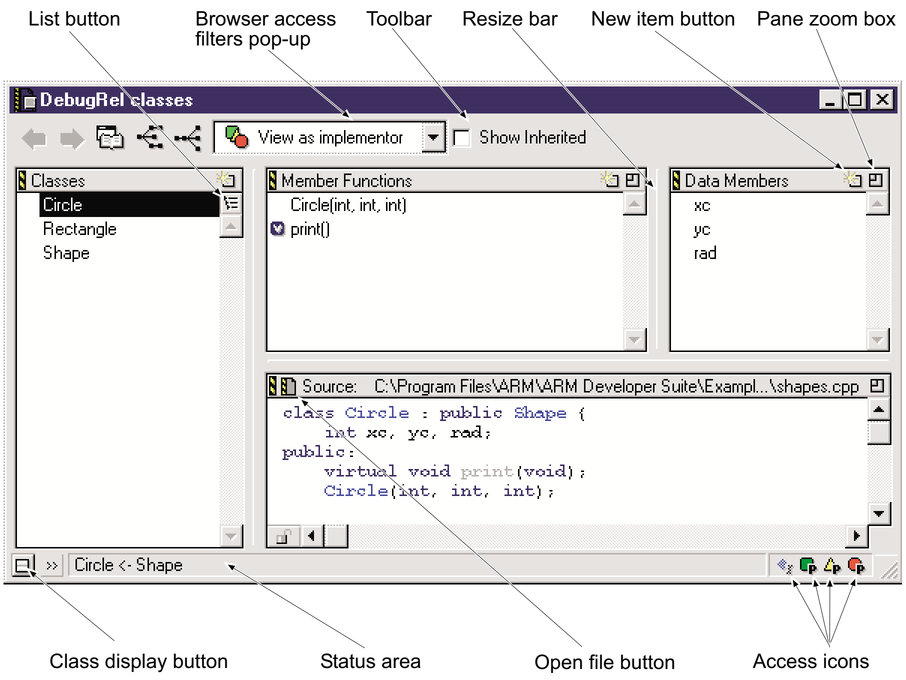
————————

Opening a Class browser window

To open the Class browser window:

1. Ensure that the browser is activated. See *Activating the browser* on page 7-5 for more information.
2. Select **New Class Browser** from the **Windows** 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 either a Single-class hierarchy window or a Multi-class hierarchy window.

   The CodeWarrior IDE displays a Class browser window (Figure 7-3 on page 7-9).



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

Figure 7-3 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 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 7-25 for more information.

Browser Access Filters pop-up menu

Use this menu to filter the display of member functions and data members. See *Filtering members by access type* on page 7-15 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 7-12 for more information.

List button Click this button to toggle between an alphabetical list or a hierarchical list in the Classes pane. See *Viewing class and member information* on page 7-12 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.

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 7-2 describes the icons.

Table 7-2 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 pop-up menu

The VCS pop-up menu is available if you have a version control system installed. See Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information.

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 7-8 for more information. The class and member information is displayed in the panes of the Class browser window. The Classes pane (Figure 7-4 on page 7-12) 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.

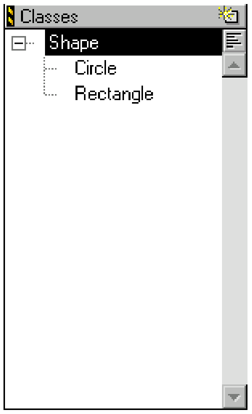


Figure 7-4 The Classes pane

2. Click the **List** button at the top right of the classes pane (see Figure 7-3 on page 7-9) 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 7-4 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.

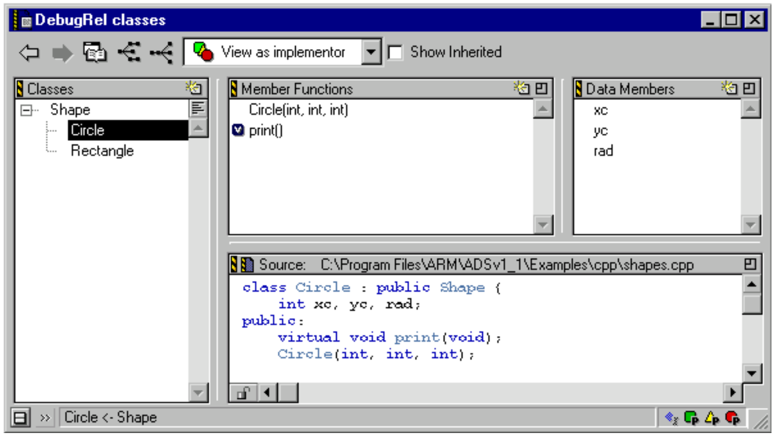
   ————————

   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 7-5 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);
    Circle(int, int, int);
```

Figure 7-5 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 7-15
   - *Opening another view from the Class browser view* on page 7-16
   - *Using the Browser Contextual menu* on page 7-22
   - *Finding declarations, definitions, overrides, and multiple implementations* on page 7-25
   - *Editing code in the browser* on page 7-29.

Filtering members by access type

You can use the **Access Filters** pop-up menu to filter the display of member functions and data members in the Class browser view. The pop-up menu commands filter the display according to public, private, and protected access types. To filter the display of members:

1. Open a Class browser window. See *Opening a Class browser window* on page 7-8 for more information.
2. Select the class you want to display. See *Viewing class and member information* on page 7-12 for more information.
3. Click on the Access Filters pop-up menu in the Class browser toolbar (Figure 7-6).

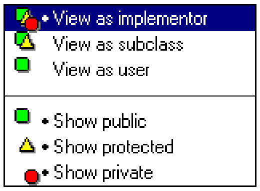


Figure 7-6 Access filters pop-up menu

The pop-up menu 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 pop-up menu:

View as implementor

Select this option to show members with public, private, and protected access.

View as subclass

Select this option to show members with public and protected access

View as user

Select this option to show only members with public access.

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 7-7).

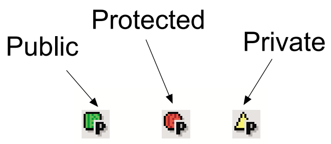


Figure 7-7 Browser access filter icons

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 multi-class hierarchy button to open a multi-class hierarchy window.
- Click the show single-class hierarchy button to open a single-class window.
- Right-click any browser symbol in the window and use the **Browser Contextual** pop-up menu. See *Using the Browser Contextual menu* on page 7-22 for more information.

Saving a default Class browser window

You can save Class browser window configurations to be used as the default for new Class browser windows. You can save:

- The size and placement of Class browser window.
- 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 7-8 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 projects.

7.3.2 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 *Activating the browser* on page 7-5 for more information.
2. Select **Browser Contents** from the **Windows** menu. The CodeWarrior IDE displays the Contents window (Figure 7-8 on page 7-17). Alternatively you can click the **Contents view** button in the Class browser toolbar, or use the **Browser Contextual** pop-up menu.

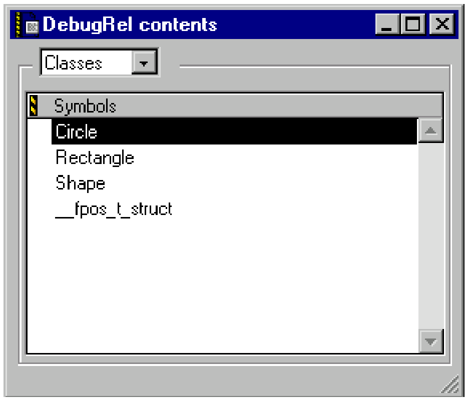


Figure 7-8 A Contents window

3. Select the category of data you want to view from the Category pop-up menu at the top of the window. The Symbols pane displays an alphabetical list of all symbols in the current build target for the selected category.

   Note

   Functions are listed alphabetically by function name, but the class name is displayed first. Therefore, it might appear that the functions are not listed alphabetically.

4. From the contents window you can:
   - Right-click on any item in the contents list to display a **Browser Contextual** pop-up menu for that item. See *Using the Browser Contextual menu* on page 7-22 for more information.
   - Double-click on any item in the contents list to open an editor window with the source code for the item.

7.3.3 Viewing class hierarchies and inheritance with the hierarchy view

You can use the browser hierarchy view to analyze inheritance in your source code. You can display a hierarchy view in two types of window:

Multi-class hierarchy window

The Multi-class 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.

Single-class hierarchy window

The Single-class hierarchy window displays a complete graphical map for a single class in the browser database. The map displays *all* immediate ancestors of the class, and all its descendants. The Multi-class hierarchy window shows only one base class.

Multi-class hierarchy window

To open a multi-class hierarchy window and view the class hierarchy for the current build target:

1. Select **Class Hierarchy Window** from the **Window** menu. The CodeWarrior IDE displays a Multi-class hierarchy window for the current build target (Figure 7-9 on page 7-19).

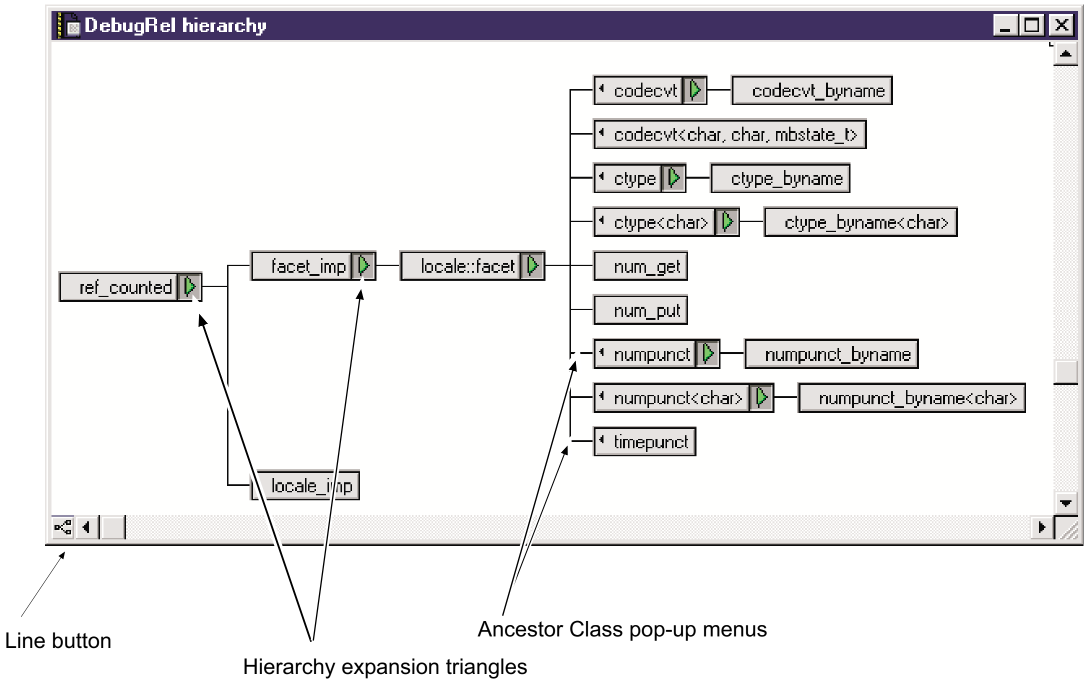


Figure 7-9 The Multi-class hierarchy window

In addition to the entry for each class, the main components of the Multi-class 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 pop-up menu

If a class has multiple base classes the hierarchy window displays a small triangle (the Ancestor Class triangle) to the left of the class name. Click on the Ancestor Class triangle, to display the **Ancestor Class** pop-up menu. Figure 7-10 shows an example.

Select the ancestor class you want from the pop-up menu to jump to the hierarchy view for the ancestor class.

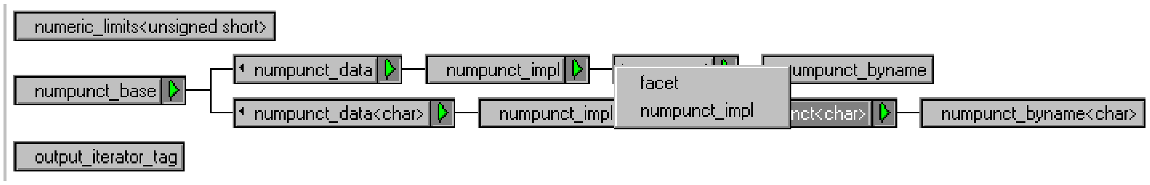


Figure 7-10 Ancestor pop-up menu

2. 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 Multi-class hierarchy window changes when you select a class in the Classes pane of the Class browser window.

3. 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 7-8 for more information.
   - Right-click on the class to open a **Browser Contextual** pop-up menu. See *Using the Browser Contextual menu* on page 7-22 for more information.

Single-class hierarchy window

The Single-class hierarchy window displays a graphical map for a single class in the browser database. To open a Single-class hierarchy view:

- Use the **Browser Contextual** menu in the Contents, Multi-class hierarchy, or Class browser window. See *Using the Browser Contextual menu* on page 7-22 for more information.
- Click the **Show Hierarchy Window** button in the browser toolbar (see Figure 7-3 on page 7-9).

Figure 7-11 on page 7-21 shows an example of the single-class hierarchy window, displaying multiple base classes and subclasses. The underlined class name is the focus of the window.

The Single-class hierarchy window works in the same way as the Multi-class hierarchy window. See *Multi-class hierarchy window* on page 7-18 for more information on using this window.

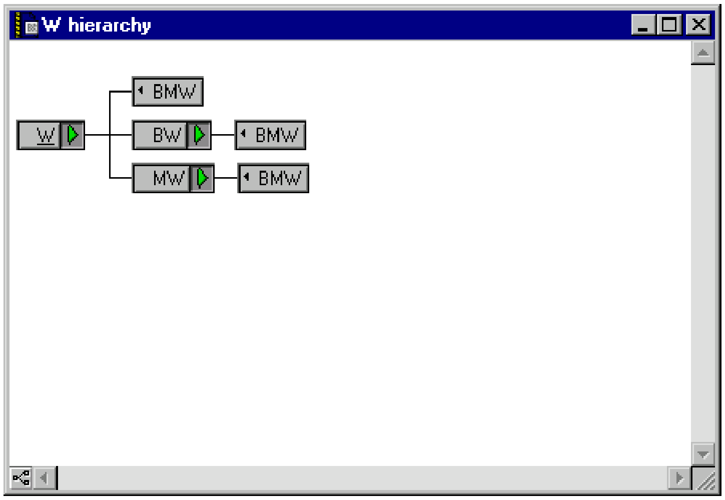


Figure 7-11 The Single-class hierarchy window

7.4 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 Contextual menu*
- *Finding declarations, definitions, overrides, and multiple implementations* on page 7-25
- *Using symbol name completion* on page 7-28
- *Editing code in the browser* on page 7-29.

7.4.1 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 pop-up menu containing a list of previous views (Figure 7-12 on page 7-22).

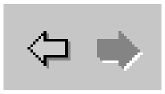


Figure 7-12 Go Back and Go Forward toolbar buttons

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

——— **Note** ———

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

———————————

7.4.2 Using the Browser Contextual menu

The **Browser Contextual** menu is a context-sensitive pop-up menu that provides quick access to browser information. The **Browser Contextual** 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 Contextual** menu:

1. Ensure that the browser is activated. See *Activating the browser* on page 7-5 for more information.

2. Open any of the Class, Contents, or Hierarchy browser windows. See *Using browser views* on page 7-8 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 Contextual** menu is displayed. The menu commands available in the menu depend on the symbol type (class, function name, enumeration, and so on), and the context in which the menu was called. Figure 7-13 shows an example of a **Browser Contextual** menu for a member function.

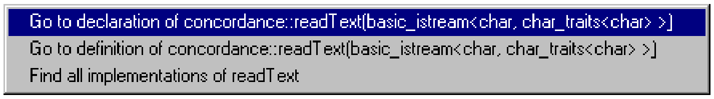

Go to declaration of concordance::readText(basic_istream<char, char_traits<char> >)
Go to definition of concordance::readText(basic_istream<char, char_traits<char> >)
Find all implementations of readText

Figure 7-13 A Browser Contextual menu for a function

For member functions, you can:

- View the function declaration. See *Viewing a class or member declaration* on page 7-25.
- View the function definition. See *Viewing a function definition* on page 7-26.
- 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 7-27.

Using the Browser Contextual 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 Single-class hierarchy window for that class.

For function names, you can use the **Browser Contextual** menu to insert function templates into your code. See *Using the Insert template commands* on page 7-24 for more information.

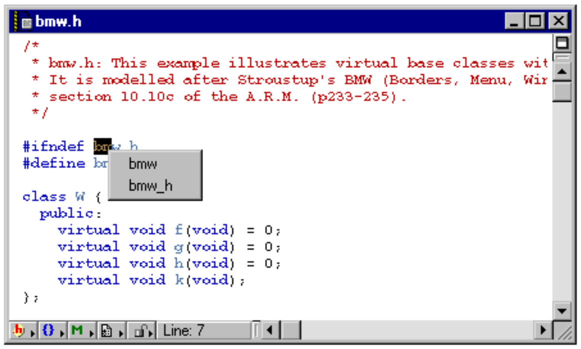
Note

- The **Browser Contextual** 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 Contextual** menu. The menu contains a list of browser symbols that match part or all of the selected text. Figure 7-14 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 classes wit
 * It is modelled after Stroustup's BMW (Borders, Menu, Wir
 * section 10.10c of the A.R.M. (p233-235).
 */

#ifndef bmw_h
#define br

class W {
  public:
    virtual void f(void) = 0;
    virtual void g(void) = 0;
    virtual void h(void) = 0;
    virtual void k(void);
};
```

Figure 7-14 Using symbol name completion

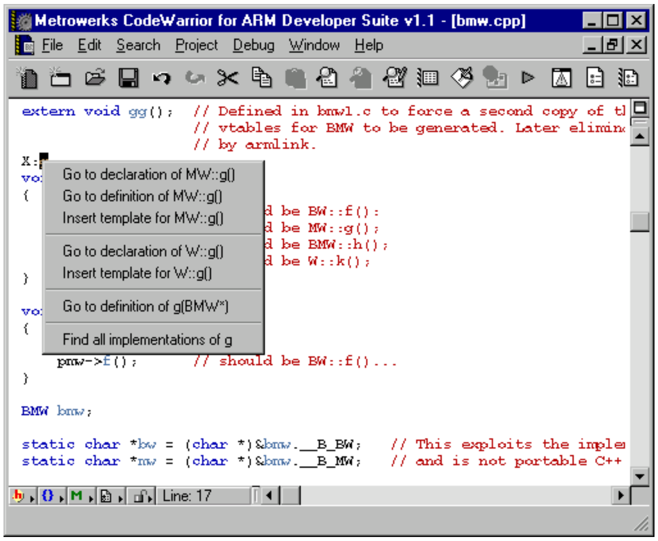
2. Select an item from the list to enter it into your text file. See *Using symbol name completion* on page 7-28 for other ways to type browser items.

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 *Browser Display* on page 8-14 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 Contextual** menu displays Insert commands for each definition. Figure 7-15 shows an example.


Metrowerks CodeWarrior for ARM Developer Suite v1.1 - [bmw.cpp]
File Edit Search Project Debug Window Help
Go to declaration of MW::g()
Go to definition of MW::g()
Insert template for MW::g()
Go to declaration of W::g()
Insert template for W::g()
Go to definition of g(BMW*)
Find all implementations of g
Line: 17

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

X:
vo:
{
                                          d be BW::f():
                                          d be MW::g();
                                          d be BMW::h();
                                          d be W::k();
}

vo:
{
    pmw->f();           // should be BW::f()...
}

BMW bmw;

static char *bw = (char *)&bmw.__B_BW;    // This exploits the imple
static char *mw = (char *)&bmw.__B_MW;    // and is not portable C++
```

Figure 7-15 Inserting a function template

3. Select the function template you want to insert. The CodeWarrior IDE inserts template code for the function.

7.4.3 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 7-26
- *Finding overrides and multiple implementations of a function* on page 7-27.

Viewing a class or member declaration

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

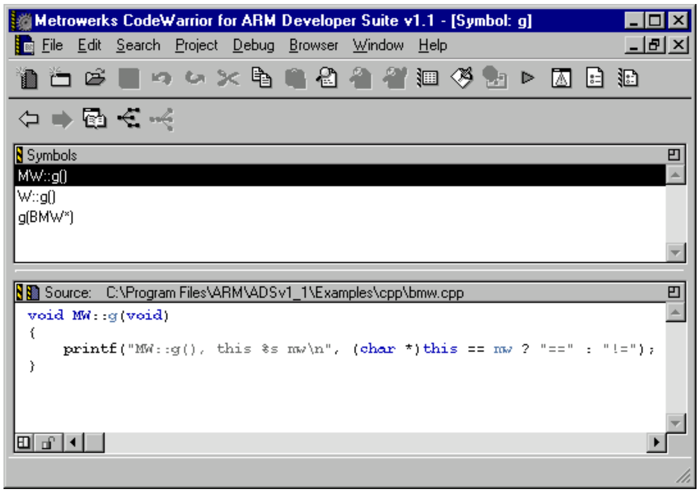
- Select a class name or data member name in a Class browser window. The declaration is displayed in the Source pane. Double-click the name to open the file that contains the declaration (if you select or double-click a function name, the function definition is displayed).
- Click the **Class Declaration** 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 Contextual** 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 Contextual** 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 7-16 shows an example for the symbol `f`.


Metrowerks CodeWarrior for ARM Developer Suite v1.1 - [Symbol: g]
File
Edit
Search
Project
Debug
Browser
Window
Help
Symbols

```
MW::g()
W::g()
g(BMW*)
```

Source: C:\Program Files\ARM\ADSv1_1\Examples\cpp\bmw.cpp

```
void MW::g(void)
{
    printf("MW::g(), this %s mw\n", (char *)this == mw ? "==" : "!=");
}
```

Figure 7-16 Symbol window for a multiply defined function

- Right-click on a class name in any browser window. The **Browser Contextual** 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

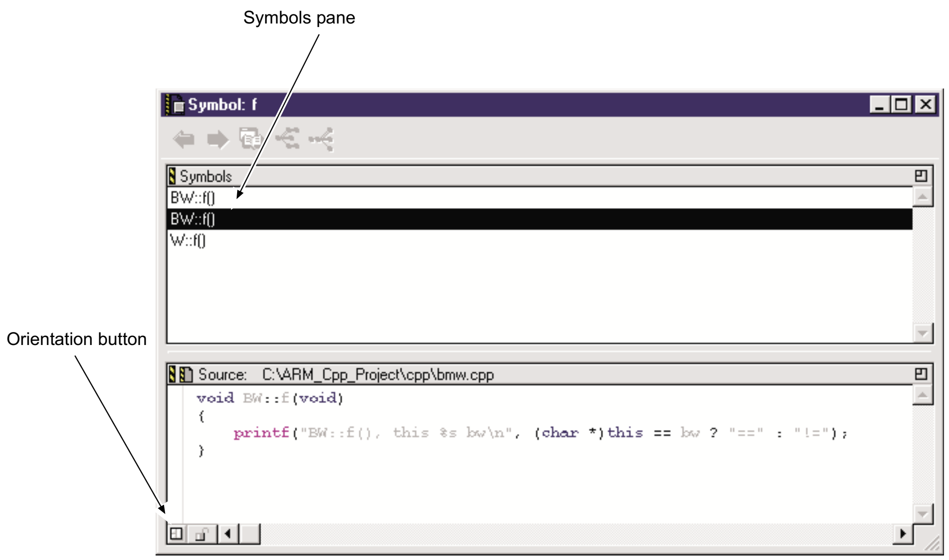
The Symbol window lists all implementations of any symbol that has multiple definitions. Typically, these symbols are multiple versions of overridden functions. However, the Symbol window works for any symbol that has multiple definitions in the browser database.

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 Contextual** menu is displayed.
3. Select **Find all implementations of** ***symbol_name*** from the **Browser Contextual** menu. The CodeWarrior IDE displays the Symbol window with a list of all definitions for the symbol (Figure 7-17 on page 7-28).

   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.



BW::f()
BW::f()
W::f()

```
void BW::f(void)
{
    printf("BW::f(), this %s bw\n", (char *)this == bw ? "==" : "!=");
}
```

Figure 7-17 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 7-8 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.

7.4.4 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 Contextual** menu to display. The menu displays a list of matching items. Select an item to enter it into your text. See *Using the Browser Contextual menu* on page 7-22 for more details.

7.4.5 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 5 *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 7-3 on page 7-9).
- Right-click on a symbol used in the source file and use the **Browser Contextual** pop-up menu to open the file.

- Type Ctrl-` to move between a source file and its corresponding header file.

7.5 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 7-36.

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 7-39.

Note

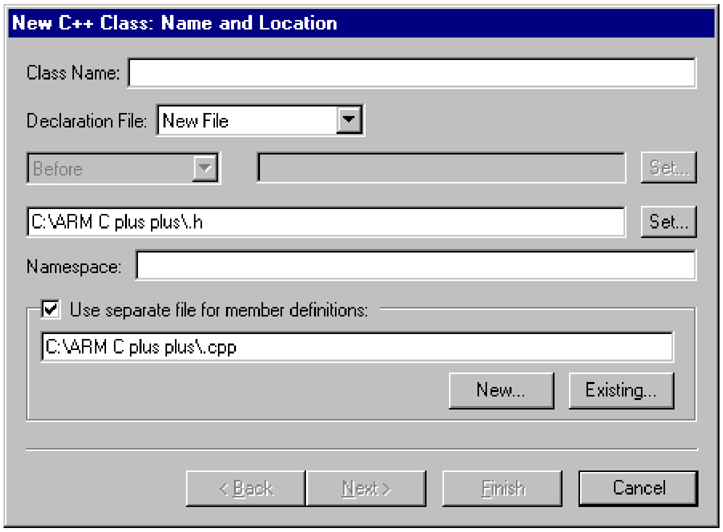
The **New Property…**, **New Method…**, **New Event Set…**, and **New Event…** menu items are not implemented by CodeWarrior for the ARM Developer Suite.

7.5.1 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 7-8 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 7-18).



Declaration File: New File

C:\ARM C plus plus\.h

Use separate file for member definitions: ✓

C:\ARM C plus plus\.cpp

Figure 7-18 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 pop-up menu to specify the type of declaration file. Depending on the option you choose, different fields become enabled below the **Declaration File** pop-up menu. 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 pop-up menu to place the new class **Before** or **After** the selected class declaration.

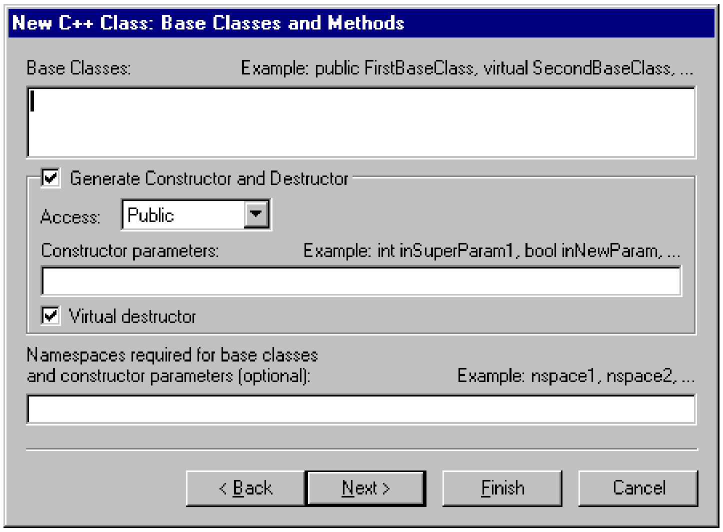
Namespace

Namespaces are not supported by the ARM and Thumb C++ compilers. 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 7-19 on page 7-33).


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 7-19 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 pop-up menu.

Constructor parameters

Enter a list of parameters for the constructor. Example parameters are listed above the field.

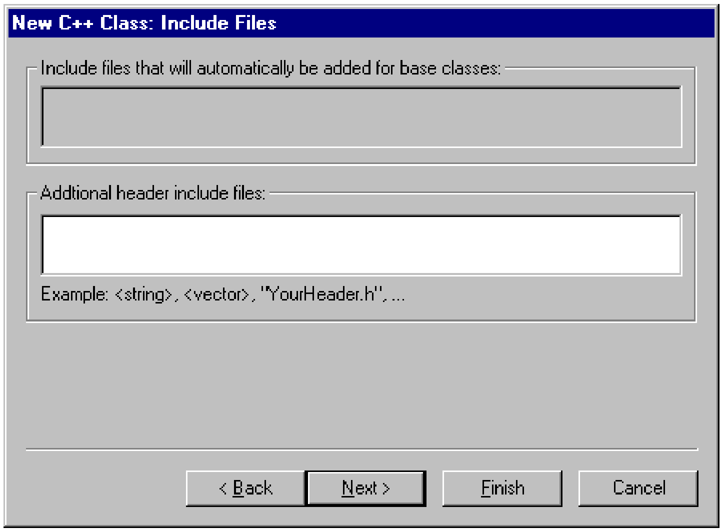
Virtual destructor

Select this checkbox to create a virtual destructor for the new class.

Namespaces

Namespaces are not supported by the ARM and Thumb C++ compilers. Leave this field empty.

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 7-20 on page 7-34).


New C++ Class: Include Files

Include files that will automatically be added for base classes:

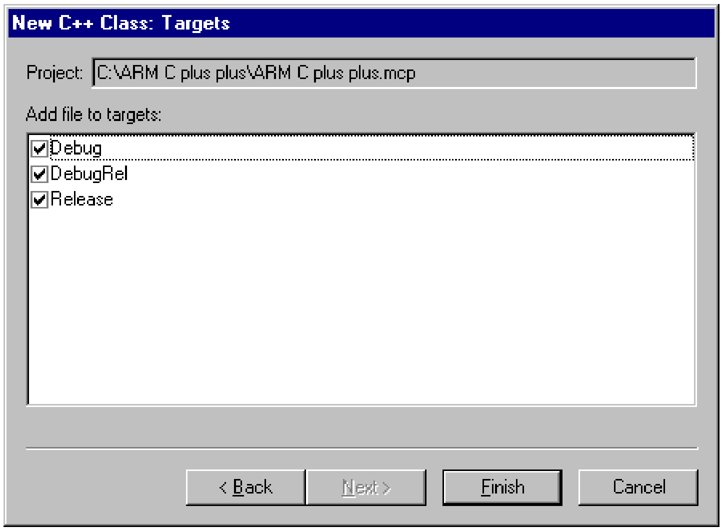
Addtional header include files:

Example: <string>, <vector>, "YourHeader.h", ...

Figure 7-20 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 7-20 on page 7-34).



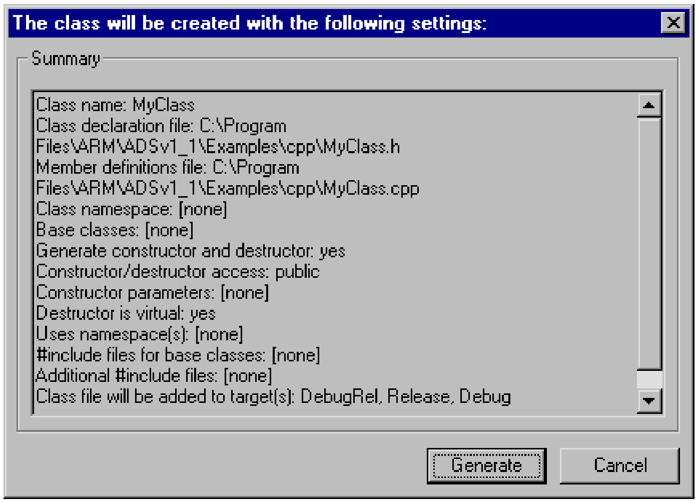
Project: C:\ARM C plus plus\ARM C plus plus.mcp

Add file to targets:

- [x] Debug
- [x] DebugRel
- [x] Release

Figure 7-21 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 7-22 on page 7-35).



```
Class name: MyClass
Class declaration file: C:\Program
Files\ARM\ADSv1_1\Examples\cpp\MyClass.h
Member definitions file: C:\Program
Files\ARM\ADSv1_1\Examples\cpp\MyClass.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): DebugRel, Release, Debug
```

Figure 7-22 New class summary

11. Click **Generate** to create the new class.

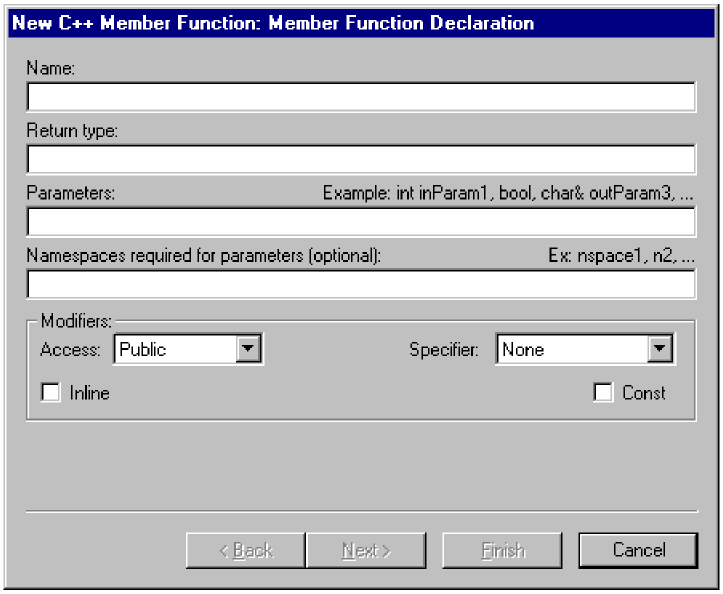
7.5.2 Using the New Member Function wizard

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 7-8 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 7-18 on page 7-32).


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 7-23 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)

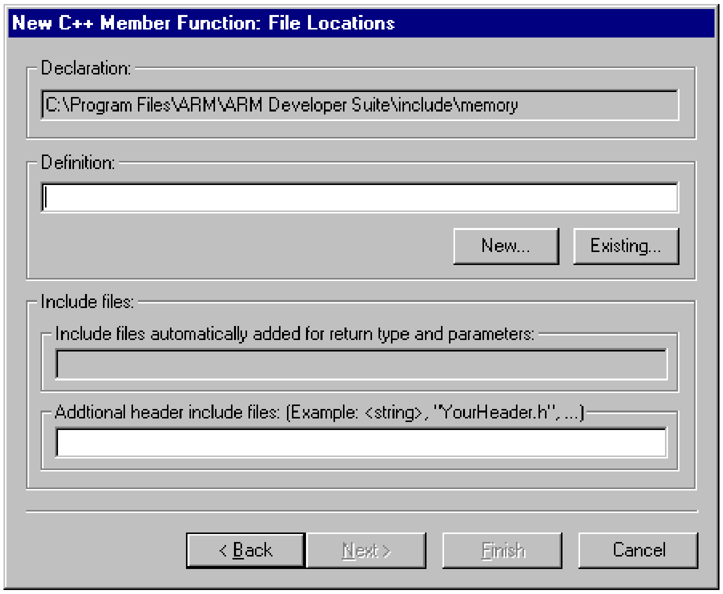
Namespaces are not supported by the ARM and Thumb C++ compilers. Leave this field empty.

Modifiers

Modify the function declaration, as required:

- Use the **Access** pop-up menu to specify whether the new member function is Public, Protected, or Private.
- Use the **Specifier** pop-up menu 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 7-24).


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

C:\Program Files\ARM\ARM Developer Suite\include\memory

Figure 7-24 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.

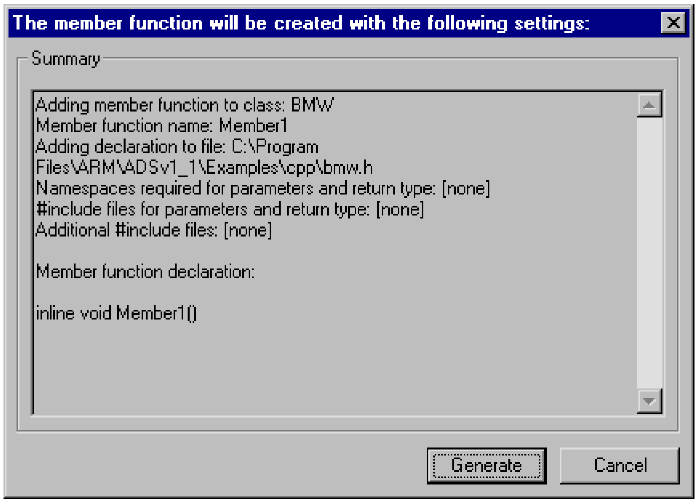
Include files automatically added…

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 7-25).



The member function will be created with the following settings:
Summary
Generate
Cancel

```
Adding member function to class: BMW
Member function name: Member1
Adding declaration to file: C:\Program
Files\ARM\ADSv1_1\Examples\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()
```

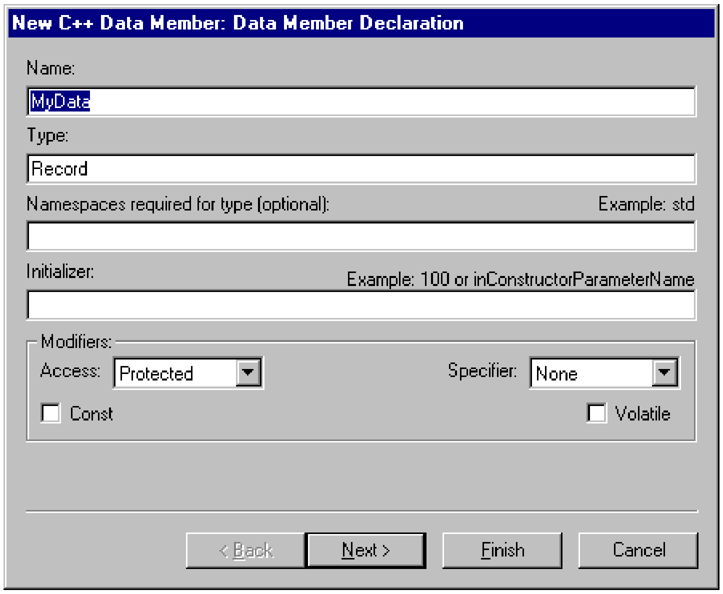
Figure 7-25 New member function declaration summary

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.

7.5.3 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 7-8 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 Name and Location page of the New Class Wizard (Figure 7-18).



New C++ Data Member: Data Member Declaration

Name: MyData

Type: Record

Namespaces required for type (optional): Example: std

Initializer: Example: 100 or inConstructorParameterName

Modifiers:

Access: Protected

Specifier: None

☐ Const

☐ Volatile

Figure 7-26 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

Namespaces are not supported by the ARM and Thumb C++ compilers. Leave this field empty.

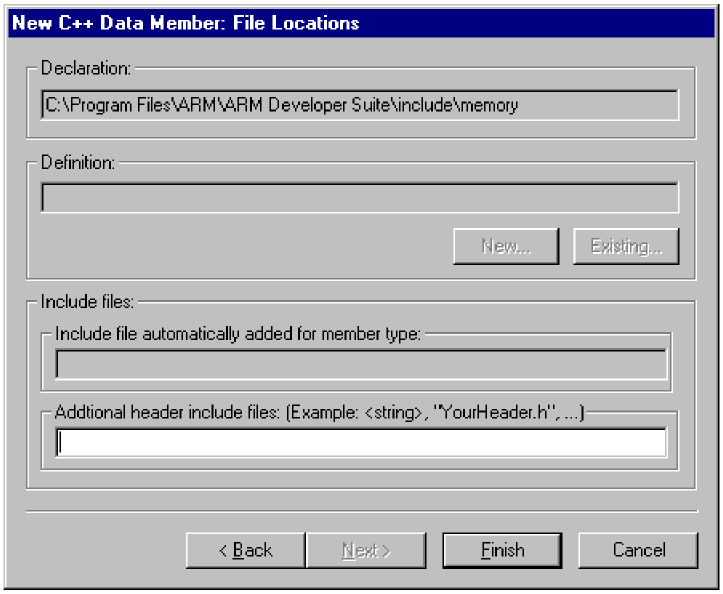
Initializer

Type an initial value for the data member. This field is optional.

Modifiers

Use the **Access** and **Specifier** pop-up menus to select the access level and member specifier for the new data member. Possible access levels include **Public**, **Protected**, and **Private**. Possible specifiers include **None**, **Static**, and **Mutable**. Enable the **Const** or **Volatile** checkboxes as desired to further describe the data member's modifiers.

5. Click **Next…** to move to the next page of the New Data Member wizard. The File Locations page is displayed (Figure 7-27).


New C++ Data Member: File Locations

Declaration: C:\Program Files\ARM\ARM Developer Suite\include\memory

Definition:

Include files:

Include file automatically added for member type:

Addtional header include files: (Example: <string>, "YourHeader.h", ...)

Figure 7-27 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.

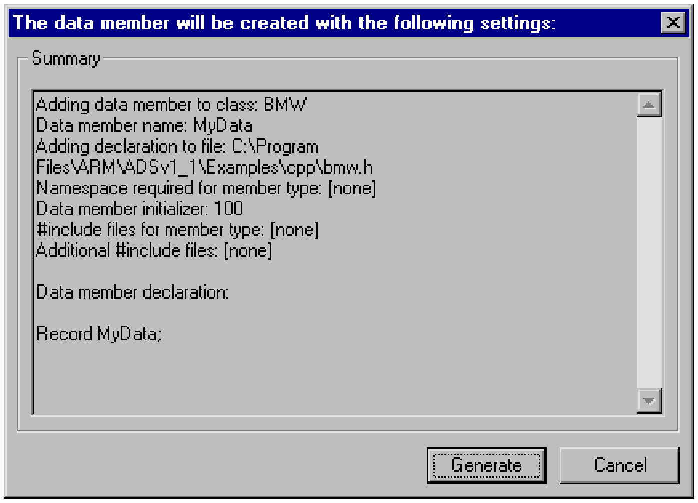
Include file automatically added for member type

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 7-28).


The data member will be created with the following settings:
Summary
Generate
Cancel

```
Adding data member to class: BMW
Data member name: MyData
Adding declaration to file: C:\Program
Files\ARM\ADSv1_1\Examples\cpp\bmw.h
Namespace required for member type: [none]
Data member initializer: 100
#include files for member type: [none]
Additional #include files: [none]

Data member declaration:

Record MyData;
```

Figure 7-28 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 8
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:

8.1 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. In addition you can customize toolbars and commands to fit 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. See *Choosing general preferences* on page 8-6 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 8-14 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 8-26 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 8-37 for more information.

8.2 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.

8.2.1 Using the IDE Preferences window

This section gives basic information on using the IDE Preferences window to configure preferences for all your CodeWarrior projects.

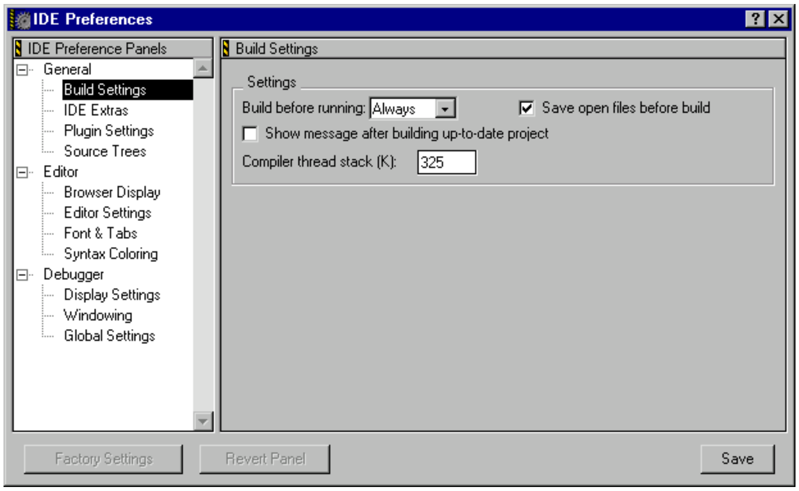
Opening the IDE Preferences panel

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 8-1 shows an example.

   Note

   The Debugger preferences panels are not used by CodeWarrior for the ARM Developer Suite.



Build Settings

Settings

Build before running: Always

[✓] Save open files before build

[ ] Show message after building up-to-date project

Compiler thread stack (K): 325

Figure 8-1 Selecting a preference 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.

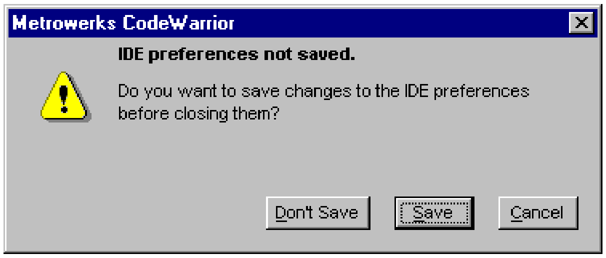
Each panel contains related options that you can set. The options you select apply to all CodeWarrior IDE projects.

3. Select the options you require. See the following sections in this chapter for detailed descriptions of the options in each configuration panel.

4. Save or discard your changes, as required. See *Saving or discarding changes* on page 8-4 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 8-2.



Metrowerks CodeWarrior

IDE preferences not saved.

Do you want to save changes to the IDE preferences before closing them?

Figure 8-2 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:

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 Panel

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.

Save Click this button to commit any changes you have made in any of the panels. If you have changed an option that requires that the project be recompiled, CodeWarrior displays a confirmation dialog box. Click **OK** or **Cancel** depending on whether you want to keep your changes or not.

8.3 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 projects.

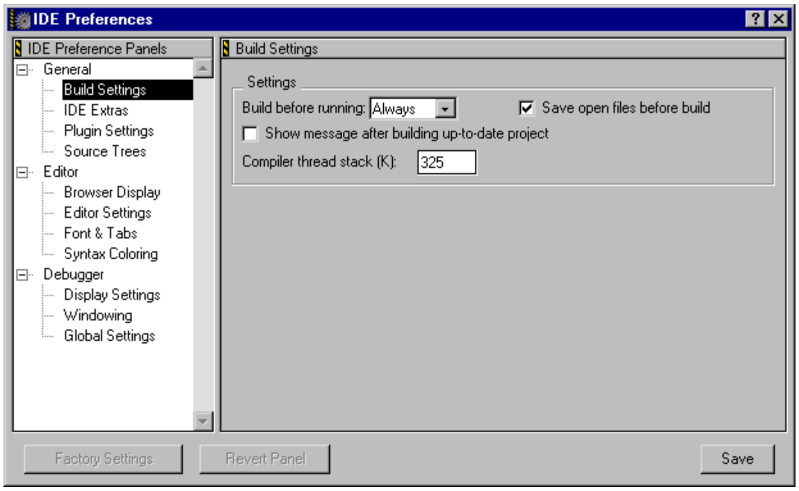
This section describes:

- *Configuring build settings*
- *Configuring IDE extras* on page 8-7
- *Configuring plug-in settings* on page 8-10
- *Configuring global source trees* on page 8-11.

8.3.1 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 8-3).



Build before running: Always
[✓] Save open files before build
[ ] Show message after building up-to-date project
Compiler thread stack (K): 325

Figure 8-3 Build Settings preference panel

2. Change the following options, as required:

   Build before running

   Use this pop-up menu to configure how the CodeWarrior IDE responds if you try to run a project and the source for the project has been changed since the last build. You can choose:

Always	Always build changed projects before running them.
Never	Never build changed projects before running them.
Ask	CodeWarrior will ask you how to proceed if you have changed the project since the last build.

   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.

   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**, or **Run** command is executed.

   Compiler thread stack

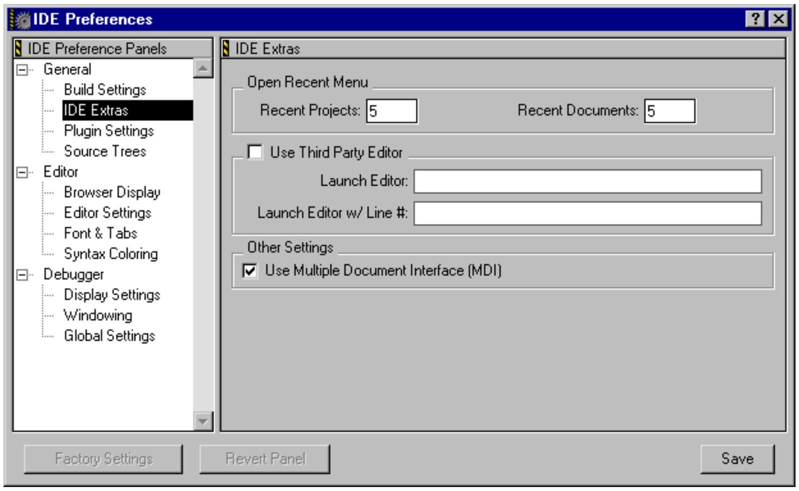
   This option is not used by CodeWarrior for the ARM Developer Suite.

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

8.3.2 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. CodeWarrior displays the IDE Extras panel (Figure 8-4).



IDE Extras

Open Recent Menu

Recent Projects: 5 Recent Documents: 5

[ ] Use Third Party Editor

Launch Editor:

Launch Editor w/ Line #:

Other Settings

[✓] Use Multiple Document Interface (MDI)

Figure 8-4 IDE Extras preference panel

There are three groups of options. For details of how to change the options see:

- *Configuring the Open Recent submenu*
- *Using a third-party text editor*
- *Other settings* on page 8-9.

Configuring the Open Recent submenu

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 8-7).
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.

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

Using a third-party text editor

You can configure CodeWarrior 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 8-7).
2. Select the **Use Third Party Editor** checkbox. When this checkbox is selected, CodeWarrior 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`	CodeWarrior expands this into the full pathname of the file.
`%line`	CodeWarrior 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 9-36 for more information.

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

Other settings

The Other Settings group has a single option that enables you to configure which Windows interface style is used by the CodeWarrior IDE. To change the Windows interface style:

1. Open the IDE Extras panel (see *Configuring IDE extras* on page 8-7).

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 **Save** to save your changes. You must quit and restart the CodeWarrior IDE to apply your changes.

8.3.3 Configuring plug-in settings

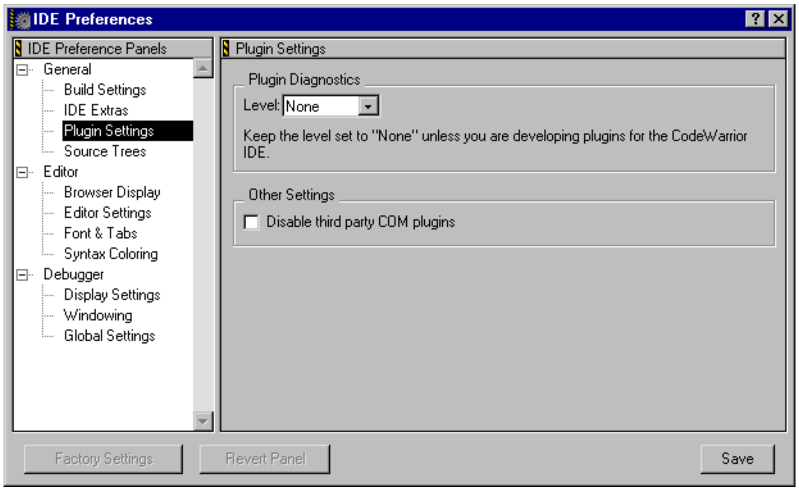
Use the Plug-in Settings panel to specify how much plug-in diagnostic information the CodeWarrior IDE provides. Plug-in diagnostic information is useful if you are using CodeWarrior to develop plug-ins for the CodeWarrior IDE. Use this panel if you have problems getting your plug-in to function properly, or if you want more information about the properties of installed plug-ins.

Note

You cannot develop CodeWarrior plug-ins with the ARM tool chain. However, if you develop a CodeWarrior plug-in using the standard Metrowerks CodeWarrior development environment you can use this option to diagnose problems when you run the plug-in from the ARM 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 8-5 on page 8-10).



Level: None

Keep the level set to "None" unless you are developing plugins for the CodeWarrior IDE.

☐ Disable third party COM plugins

Figure 8-5 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, compilers, 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 **Save** to save your settings. CodeWarrior warns that you must quit and restart the CodeWarrior IDE for the changes to take effect. Plug-in diagnostics are generated when you restart CodeWarrior.

8.3.4 Configuring global source trees

The Source Trees settings panel enables you to define global 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. See *Configuring access paths* on page 9-20 for more information.

You can define 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 9-45 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 8-6 on page 8-12). The source trees panel displays a list of currently-defined source paths.

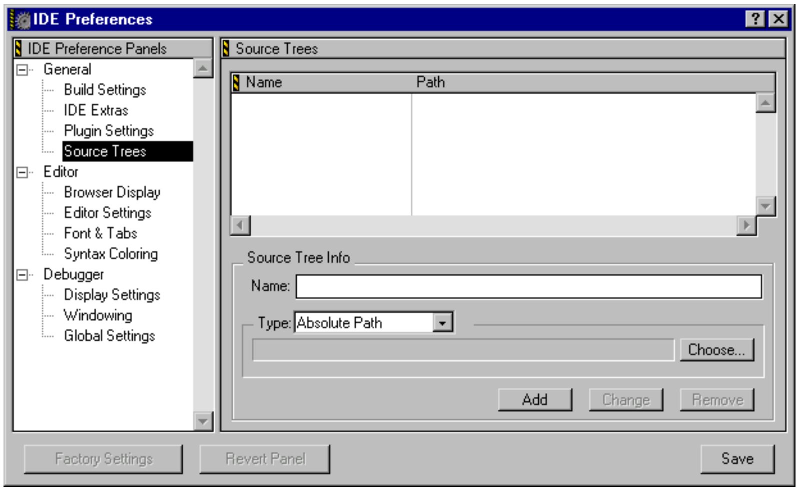


Figure 8-6 Source Trees panel

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** pop-up menu 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. 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. For example, to add the directory defined by the `ARMHOME` registry entry, enter:

     ```
     HKEY_LOCAL_MACHINE\SOFTWARE\ARM Limited\ARM Developer
     Suite\v1.1\ARMHOME
     ```

5. Click **Add** to add a new source tree, or click **Change** if you are modifying an existing source tree.

6. Click **Save** to save 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 (Figure 8-7). See *Configuring access paths* on page 9-20 for more information on adding access paths to CodeWarrior projects.

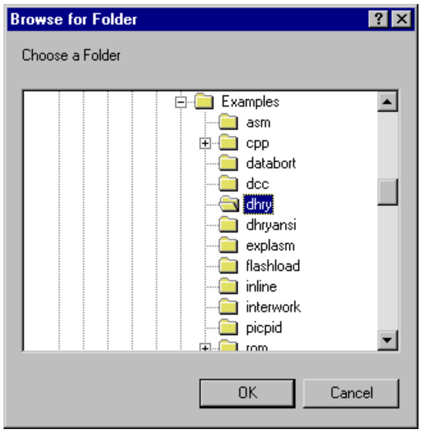


Figure 8-7 Example Source path

8.4 Choosing editor preferences

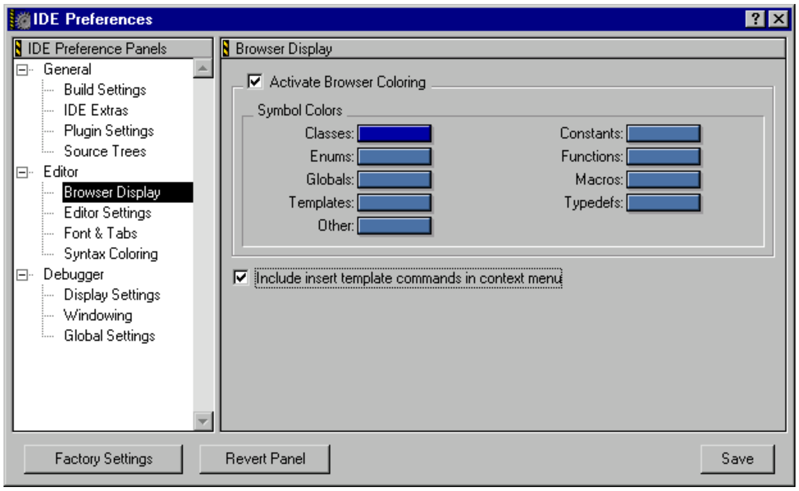
This section describes the preference panels that control editor features. The editor panels are described in:

- *Browser Display*
- *Editor settings* on page 8-15
- *Font & Tabs* on page 8-19
- *Syntax Coloring* on page 8-22.

8.4.1 Browser Display

The **Browser Display** panel enables you to customize the browser. To open the Browser Display panel:

1. Select **Preferences…** from the **Edit** menu. CodeWarrior displays the IDE Preferences dialog.
2. Click **Browser Display** in the IDE Preference Panels list. CodeWarrior displays the Browser Display panel (Figure 8-8).



Browser Display: [✓] Activate Browser Coloring; [✓] Include insert template commands in context menu

Figure 8-8 Browser Display options

Use the Browser Display window to specify how the browser interacts with other parts of the CodeWarrior IDE. For more information see:

- *Setting browser coloring options* on page 8-15
- *Including insert template commands in the context menu* on page 8-15.

Setting browser coloring options

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. To set this option:

1. Open the Browser Display window (see *Browser Display* on page 8-14 for details).
2. Select **Activate Browser Coloring**. The color choice for each symbol type is displayed in both the editor window and the browser window.
3. Specify colors you want to use for the different symbol types. The Browser Display window displays a color sample next to each symbol type. To change the color:
    a. Click the color sample next to the symbol type you want to change. The CodeWarrior IDE displays the standard Windows color selector.
    b. Select the color you want and click **OK**. The color sample changes to the new color.
4. Click **Save** to save your settings.

Including insert template commands in the context menu

You can use the browser contextual menu to insert function templates into your source code. See *Using the Insert template commands* on page 7-24 for more information. To configure the CodeWarrior IDE to include an **Insert Template** command in the context menu:

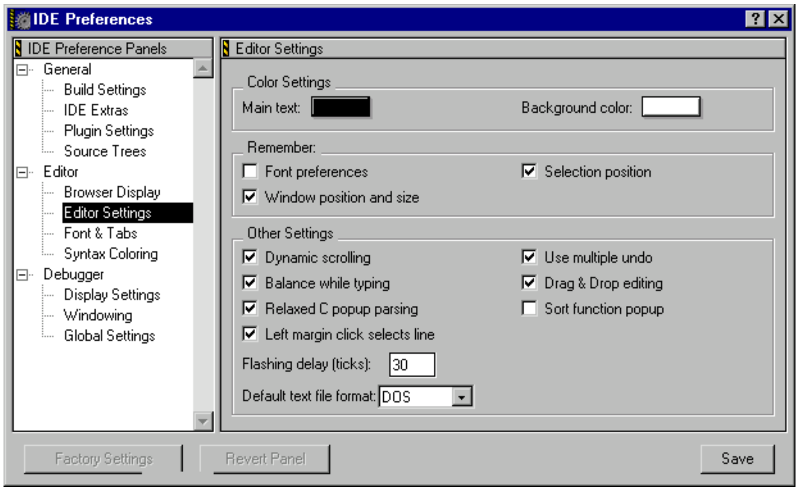
1. Open the Browser Display window (see *Browser Display* on page 8-14 for details).
2. Select the **Include insert template commands in context menu** option. The CodeWarrior IDE adds an **Insert Template** command to your context pop-up menus. Figure 7-15 on page 7-25 shows an example.
3. Click **Save** to save your settings.

8.4.2 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. CodeWarrior displays the IDE Preferences dialog.

2. Click **Editor Settings** in the IDE Preference Panels list. CodeWarrior displays the Editor Settings panel (Figure 8-9).



IDE Preferences

IDE Preference Panels
- General
  - Build Settings
  - IDE Extras
  - Plugin Settings
  - Source Trees
- Editor
  - Browser Display
  - Editor Settings
  - Font & Tabs
  - Syntax Coloring
- Debugger
  - Display Settings
  - Windowing
  - Global Settings

Editor Settings

Color Settings
Main text: ■ Background color: □

Remember:
- [ ] Font preferences
- [✓] Selection position
- [✓] Window position and size

Other Settings
- [✓] Dynamic scrolling
- [✓] Use multiple undo
- [✓] Balance while typing
- [✓] Drag & Drop editing
- [✓] Relaxed C popup parsing
- [ ] Sort function popup
- [✓] Left margin click selects line

Flashing delay (ticks): 30

Default text file format: DOS

Factory Settings | Revert Panel | Save

Figure 8-9 Editor Settings preference panel

There are three groups of options. For more information on changing the options see:

- *Specifying editor color settings*
- *Setting Remember options* on page 8-17
- *Specifying Other Settings* on page 8-17.

Specifying editor color settings

The Color Settings panel controls the main text color (non-syntax) and the background color in the editor and browser windows. To set the color options:

1. Open the Editor Settings panel (see *Editor settings* on page 8-15). The Color Settings group of options at the top of the panel displays a color sample for the Main text, and the Background color:

   Main text color

   The Main text color sample displays the color of any text not colored by the **Browser Display**, **Syntax Coloring**, or **Custom Keywords** color sets. See *Browser Display* on page 8-14 and *Syntax Coloring* on page 8-22 for more information on these color sets.

   Background color

   This color sample displays the background color of the editor and browser windows.

2. Click either the Main text color sample or the Background color sample. The CodeWarrior IDE displays the standard Windows color selector.

3. Select the color you want and click **OK**. The color sample changes to the new color.

4. Click **Save** to save and apply your settings.

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 8-15).

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 and have checked out a read-only copy of a file. See *Performing common VCS operations* on page 10-7 for more information.

3. Click **Save** to save your settings.

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 8-15).

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 Flashing Delay (see below), 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 Flashing Delay to 0.

————————

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** pop-up menu. You must deselect this option if you use non-standard macros that can interfere with K&R-styled code.

——— **Note** ———

Some macro functions will not be recognized when this option is enabled. If you encounter problems with viewing function names, disable this option and try again.

————————

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 *Redo, Multiple Undo, and Multiple Redo* on page B-6 for more information.

Drag & 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 5-13 for more information on Drag & Drop editor features.

Sort function popup

Select this option if you want items in the **Functions** pop-up menu in the editor window to be sorted alphabetically by default. See *Using the Functions pop-up menu* on page 5-17 for more information.

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.

Flashing delay

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 5-14, 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 text file format

Use the **Default text file format** pop-up menu 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 **Save** to save your changes. The new settings are applied immediately.

8.4.3 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.

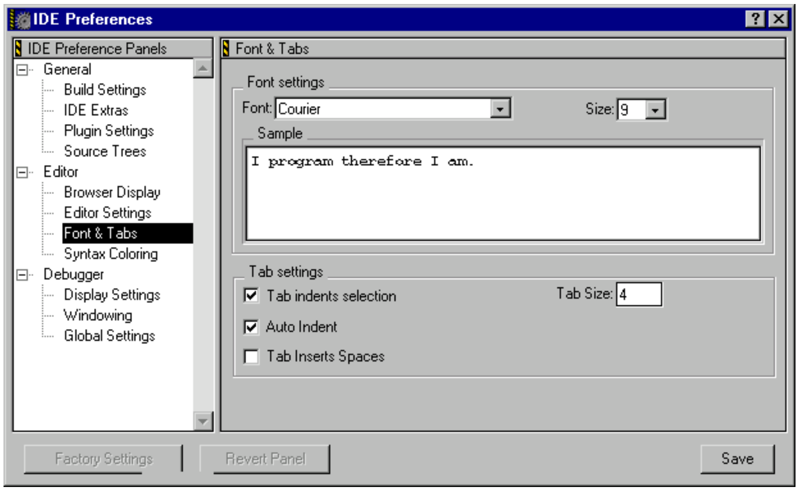
Note

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 8-17 for more information. See *Setting the font and tabs for a single file* on page 8-20 for detailed instructions.

To open the Font & Tabs panel:

1. Select **Preferences…** from the **Edit** menu. CodeWarrior displays the IDE Preferences dialog.

2. Click **Font & Tabs** in the IDE Preference Panels list. The CodeWarrior IDE displays the Font & Tabs panel (Figure 8-10).


IDE Preferences

IDE Preference Panels: General (Build Settings, IDE Extras, Plugin Settings, Source Trees); Editor (Browser Display, Editor Settings, Font & Tabs, Syntax Coloring); Debugger (Display Settings, Windowing, Global Settings)

Font & Tabs

Font settings
Font: Courier
Size: 9
Sample: I program therefore I am.

Tab settings
☑ Tab indents selection
Tab Size: 4
☑ Auto Indent
☐ Tab Inserts Spaces

Figure 8-10 Font & Tabs preference panel

For more information on setting font and tabs options see:

- *Setting the font and tabs for a single file* on page 8-20
- *Setting font and tabs defaults* on page 8-21.

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 8-17 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 8-19).

4. Select the display font and font size from the pop-up menus, 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:

   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.

   ————————

   Auto Indent

   Select this option to maintain the current indent level when you press the Enter key.

   Tab Inserts Spaces

   Select this option to insert space characters, instead of a tab character, when you press the Tab 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.

6. Click **Save** to save your changes. CodeWarrior 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 and have checked out a read-only copy of a file. See *Performing common VCS operations* on page 10-7 for more information.

   ————————

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 8-17 for more information.

When this option is not selected, any changes you make in the Font & Tabs configuration panel apply to all CodeWarrior editing sessions.

2. Follow the instructions in *Setting the font and tabs for a single file* on page 8-20 to select font and tab options. You do not require an open editor window to set default values.

8.4.4 Syntax Coloring

The Syntax Coloring preferences panel enables you to change the default color for comments, keywords, and strings in your source code.

Note

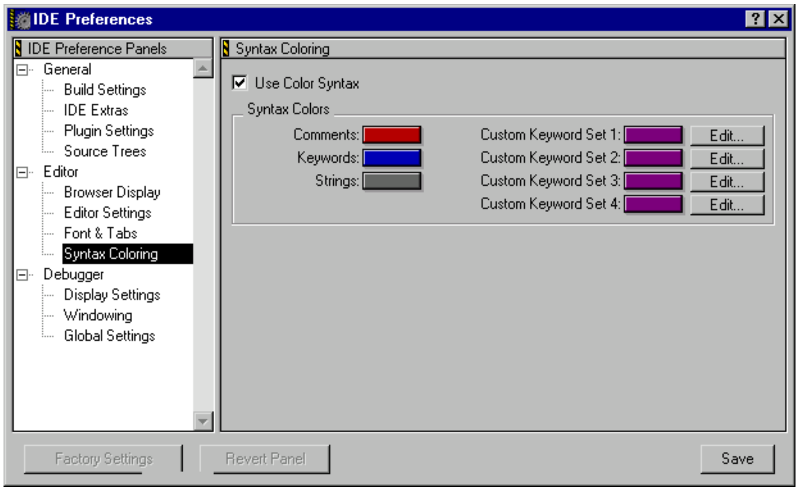
Syntax coloring does not apply to assembly language source. However, you can use browser coloring to highlight assembly language constructs in the browser.

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 Syntax Coloring panel:

1. Select **Preferences…** from the **Edit** menu. CodeWarrior displays the IDE Preferences window.

2. Click **Syntax Coloring** in the IDE Preference Panels list. CodeWarrior displays the Syntax Coloring panel (Figure 8-11).



☑ Use Color Syntax

Comments: 🟥
Keywords: 🟦
Strings: ⬛
Custom Keyword Set 1: 🟪
Custom Keyword Set 2: 🟪
Custom Keyword Set 3: 🟪
Custom Keyword Set 4: 🟪

Figure 8-11 Syntax Coloring preference panel

3. Select the **Use Color Syntax** option to turn syntax coloring on.

   Table 8-1 lists each element of text that the CodeWarrior editor displays in color.

Table 8-1 Syntax coloring highlights

Element	Description
Main text	Anything that is not a comment, keyword, or custom keyword, such as literal values, variable names, function names, and type names.
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.
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.

Changing syntax highlighting colors

The CodeWarrior IDE can use different colors for each type of text. To change the syntax highlighting:

1. Open the Syntax Coloring panel (see *Syntax Coloring* on page 8-22).
2. Click the color sample next to type of text you want to change. For example, to change the color used for comments in your source code, click the Comments color sample. CodeWarrior displays a standard system color picker dialog.
3. Select the color you want from the color picker dialog and click **OK**.
4. Click **Save** to save your changes.

Note

You can use **Syntax Coloring** from the **Document Settings** pop-up menu to turn syntax coloring on or off as you view a particular file. See *Controlling color* on page 5-16 for more information.

Using color for custom keywords

You can use the Custom Keywords dialog box 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 Syntax Coloring panel (see *Syntax Coloring* on page 8-22).
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 8-12 on page 8-24).

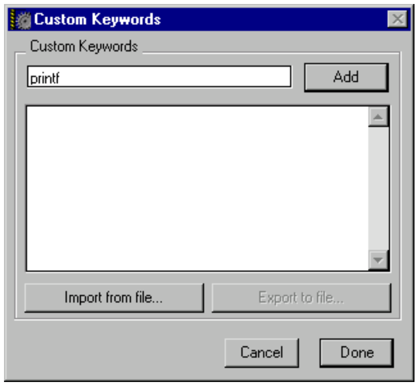


Figure 8-12 Custom Keywords dialog box

3. Type a keyword in the Add text field. You can also import sets of custom keywords that you have already saved. See *Importing and exporting custom keywords* on page 8-25 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 **Save** in the Syntax Coloring panel. 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. That is, definitions in Custom Keyword Set 1 are used before those in Custom Keyword Set 2, and so on.
- You can also set target-specific colors for custom keywords. See *Custom Keywords* on page 9-133 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 Syntax Coloring panel (see Figure 8-11 on page 8-22).
2. Click **Edit** to the left of the Custom Keyword Set you want to export or import to (see Figure 8-11 on page 8-22).
3. Click either:
   - **Export to file…**, if you want to save the current list of keywords.
   - **Import from file…**, if you want to save the current list of keywords.

   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 **Save** in the Syntax Coloring panel to apply the changes to your current editor files.

8.5 Setting commands and key bindings

This section describes how to:

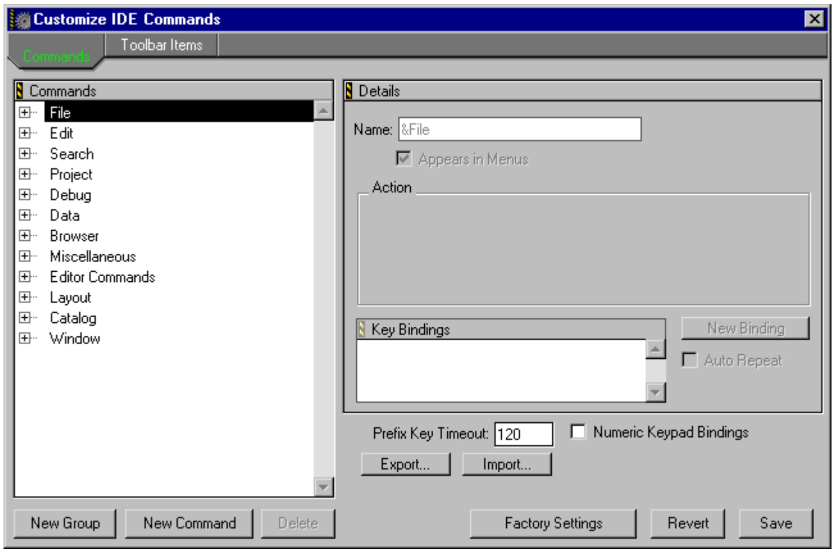
- specify customized commands that can appear in the CodeWarrior IDE menus
- assign keyboard shortcuts to commands and change keyboard shortcuts that are already defined.

See also *Customizing toolbars* on page 8-37 for information on customizing CodeWarrior IDE toolbars.

8.5.1 Opening the Customize IDE Commands window

The Customize IDE Commands window contains two tabbed panels that enable you to specify your own commands, assign keybindings to commands, and customize the CodeWarrior IDE toolbar. To open the Customize IDE commands window:

1. Select **Commands & Keybindings…** from the **Edit** menu. CodeWarrior displays the Customize IDE Commands window (Figure 8-13). The window contains two tabbed panels.



Name: &File

[✓] Appears in Menus

Prefix Key Timeout: 120

[ ] Numeric Keypad Bindings

Figure 8-13 Key Bindings panel

2. Click either:
   - The **Toolbar Items** tab, to display the toolbar control elements that you can add to the CodeWarrior IDE toolbars. See *Customizing toolbars* on page 8-37 for more information.

- 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.

3. Click a hierarchical control next to a command group to display the commands for the command group. Figure 8-14 shows an example for the Edit group of commands.

   ——— **Note** ———

   Some commands are not implemented by the ARM version of the CodeWarrior IDE.

   ————————

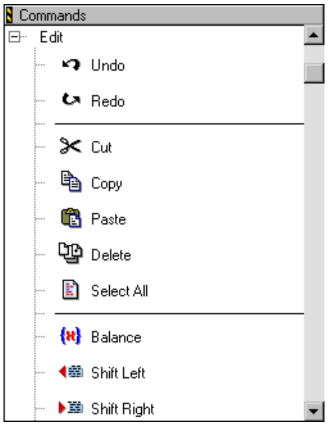


Figure 8-14 List of Edit commands

4. Click on a command to select it. The default key bindings for the command are displayed. See *Customizing keybindings* on page 8-30 for more information on modifying key bindings. See *Customizing toolbars* on page 8-37 for more information on adding commands to the CodeWarrior IDE toolbar.

8.5.2 Adding your own commands to the CodeWarrior IDE

You can use the **Commands & Keybindings** command 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:

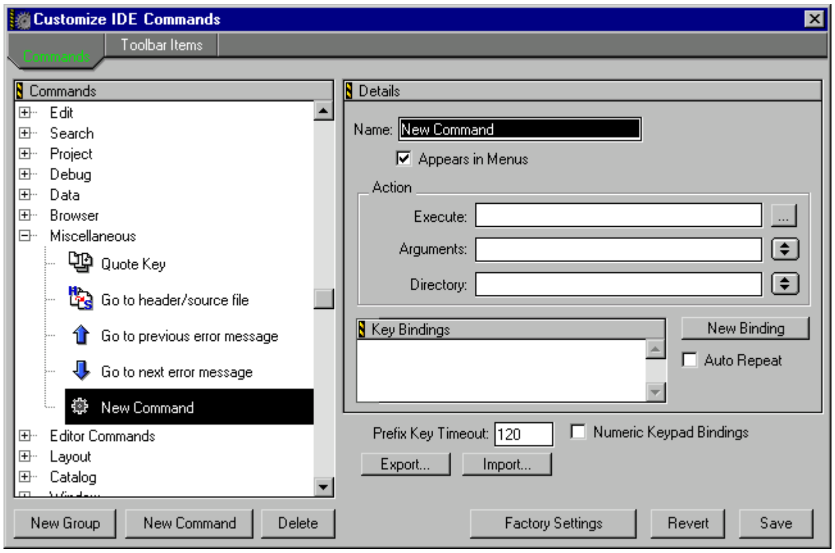
1. Open the Customize IDE Commands window. See *Opening the Customize IDE Commands window* on page 8-26.
2. Create a new group if you want to create a new menu for the command. See *Creating a new command group* on page 8-29 for more information.

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 groups.

   The CodeWarrior IDE adds Action configuration fields to the Commands window (Figure 8-15).



Name: New Command

[✓] Appears in Menus

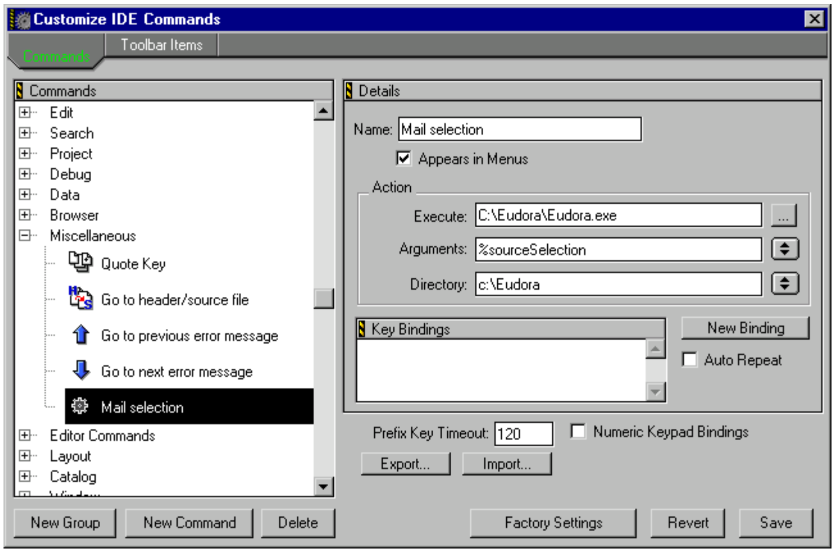
Prefix Key Timeout: 120

Figure 8-15 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 pop-up menu button next to the text field to select arguments from a list of CodeWarrior IDE internal variables.

   For example, select the **Editor Selected Text** pop-up menu item to specify the text selected in the current editor window as an argument to the command. Figure 8-16 shows an example.



Name: Mail selection
[✓] Appears in Menus
Action
Execute: C:\Eudora\Eudora.exe
Arguments: %sourceSelection
Directory: c:\Eudora
Prefix Key Timeout: 120

Figure 8-16 Mail selected command

6. Enter the name of the working directory from which the command is to be executed, if required. You can click on the pop-up menu 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** pop-up menu 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 8-30 for more information.

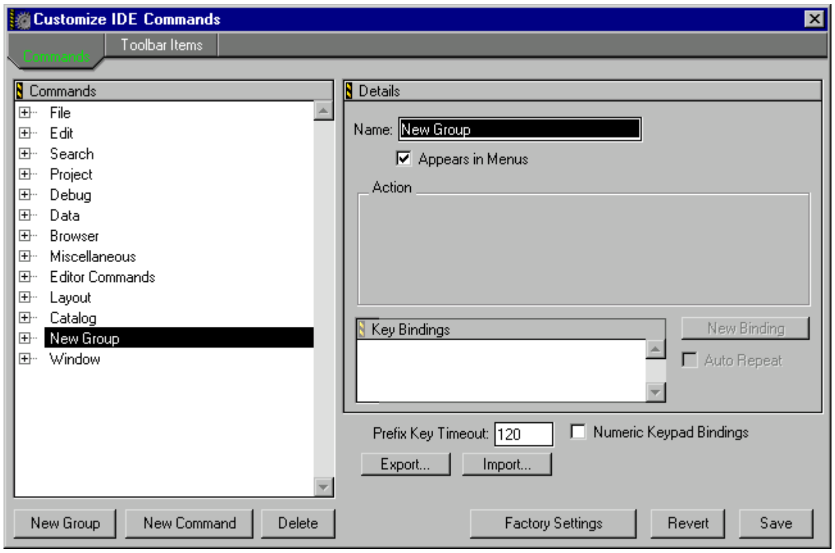
8. Click **Save** to save your settings. The CodeWarrior IDE adds the command to the specified group.

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 8-26.

2. Click **New Group…**. The CodeWarrior IDE inserts the new group into the commands list (Figure 8-17).



Name: New Group

☑ Appears in Menus

Prefix Key Timeout: 120

Figure 8-17 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 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.

8.5.3 Customizing keybindings

This section describes how to customize the default CodeWarrior 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* on page 8-31
- *Using multiple-keystroke bindings* on page 8-31
- *Setting the Prefix Key Timeout* on page 8-31
- *Using a Quote Key prefix to create single-key keybindings* on page 8-32
- *Setting Auto Repeat for keybindings* on page 8-33

- *Modifying key bindings* on page 8-34
- *Adding a new keybinding* on page 8-35
- *Deleting a keybinding* on page 8-35
- *Exporting key bindings* on page 8-36
- *Importing key bindings* on page 8-36.

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* on page 8-31).

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 8-26).

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 **Save** 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 8-26).
2. Click the hierarchical control next to the Miscellaneous group and select the Quote Key entry (Figure 8-18).

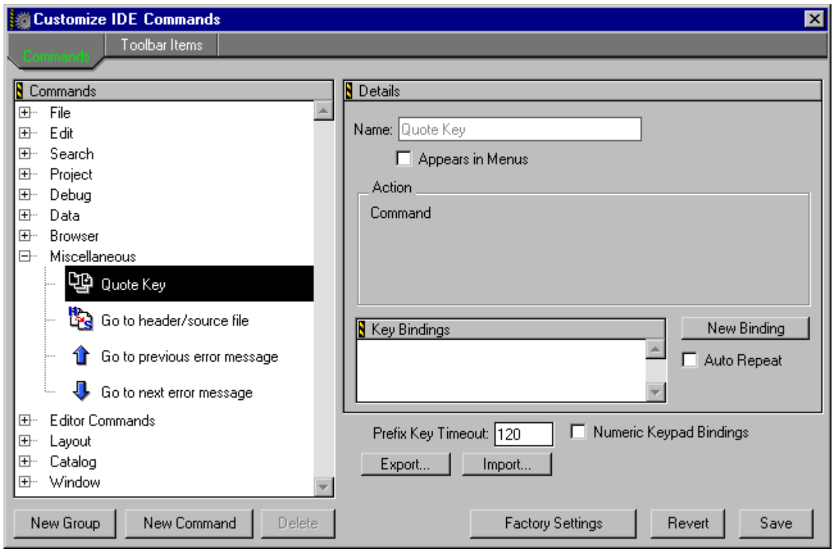


Figure 8-18 Selecting the quote key

3. Click New Binding to display the Edit Key Binding dialog (Figure 8-19).

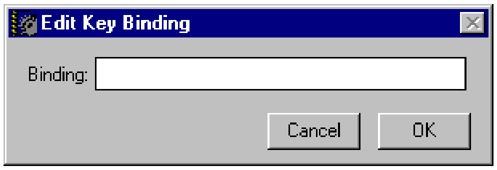


Figure 8-19 New Key binding for the Quote key

4. Type the key you want to use as the quote key prefix and click **OK** to set the key binding.

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

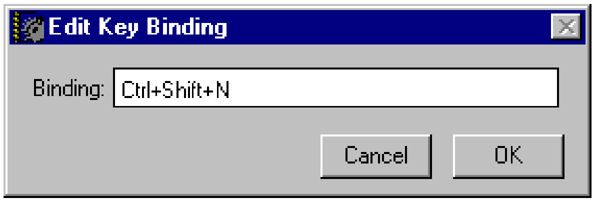
Setting Auto Repeat for keybindings

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. See *Modifying key bindings* on page 8-34 for more information.

Modifying key bindings

The CodeWarrior IDE defines default keybindings for many commands. See *CodeWarrior IDE default key bindings* on page B-25 for a list of the default keybindings. To modify the default key bindings for the CodeWarrior IDE:

1. Open the Key Bindings panel (see *Opening the Customize IDE Commands window* on page 8-26).
2. Click the hierarchical control next to the group of commands that contains the command you want to modify. Most commands are listed under groups that correspond to the menus in the CodeWarrior IDE.
3. Double-click the command you want to modify, or select it and press the Enter key. The Change Binding dialog box is displayed (Figure 8-20).



Binding: Ctrl+Shift+N

Figure 8-20 Change Key Binding dialog box

4. Press the key combination you want to use for the command. For example, press the Control key and the 8 key to make the command key Control-8.

   Note

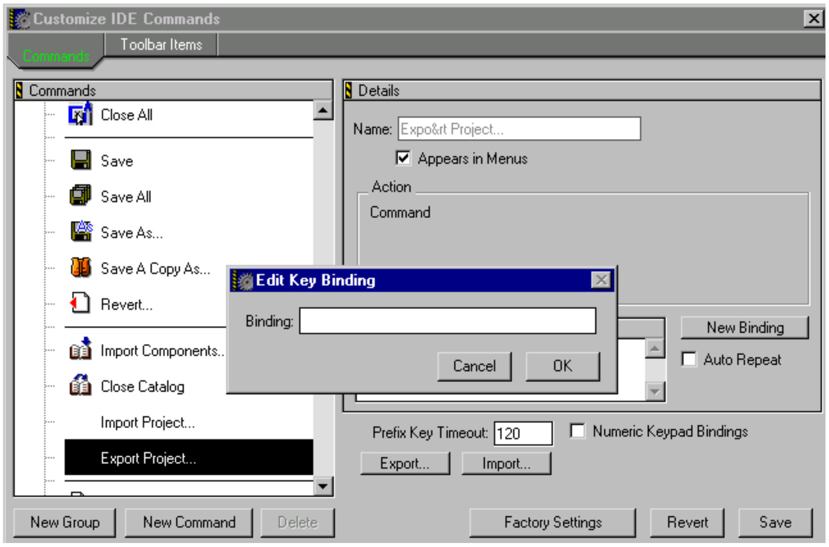
   You cannot edit the contents of the Binding field.

5. Click **OK** to set the keybinding.
6. Select:
   - **Appears in Menus** if you want the command to appear in the CodeWarrior IDE menus.
   - **Auto Repeat**, if you want the command to be applied repeatedly when you hold down its key combination.
7. Click **Save** to save your changes.

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 8-26).
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 8-21).



Name: Expo&rt Project...

[✓] Appears in Menus

Prefix Key Timeout: 120

Figure 8-21 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 **Save** 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 8-26).

2. Select the keybinding you want to delete.
3. Press the Delete key. The keybinding is deleted from the list of keybindings.
4. Click **Save** to save 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 8-26).
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 8-26).
2. Click **Import** and select the keybindings file from the standard file dialog.
3. Click **Open** to read the keybindings from the file.

8.6 Customizing toolbars

This section describes how to customize the CodeWarrior IDE toolbars. It describes:

- *Toolbar overview* on page 8-37
- *Showing and hiding a toolbar* on page 8-38
- *Modifying a toolbar* on page 8-38.

8.6.1 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 pop-up menus.

Figure 8-22 shows the default toolbar from the CodeWarrior project window.

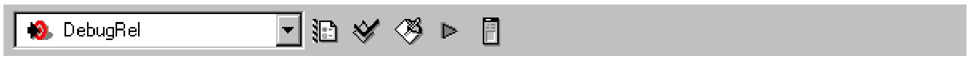


Figure 8-22 The Project window toolbar

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 8-38 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 8-42 for more information.

8.6.2 Showing and hiding a toolbar

The **Window** menu contains a **Toolbar** 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.

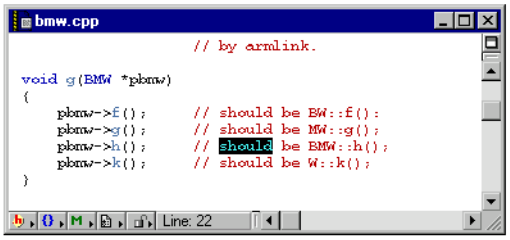
To show or hide a window toolbar:

1. Click on the window which you want to configure to make it the active window.
2. Select either:
   - **Window** → **Toolbar** → **Hide Window Toolbar** to hide the window toolbar
   - **Window** → **Toolbar** → **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 5-7 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 8-23 shows an example.



```
                    // by armlink.

void g(BMW *pbmw)
{
    pbmw->f();      // should be BW::f():
    pbmw->g();      // should be MW::g();
    pbmw->h();      // should be BMW::h();
    pbmw->k();      // should be W::k();
}
```

Figure 8-23 The editor window with hidden toolbar

8.6.3 Modifying a toolbar

You can modify a toolbar by:

- *Adding a toolbar element* on page 8-39
- *Removing a toolbar element* on page 8-41
- *Removing all toolbar elements* on page 8-41
- *Restoring a toolbar to default settings* on page 8-42.

There are restrictions on the elements you can add or remove from a toolbar. These are described in *Adding a toolbar element* on page 8-39 and *Removing a toolbar element* on page 8-41.

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 (Figure 8-24 on page 8-39). 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.

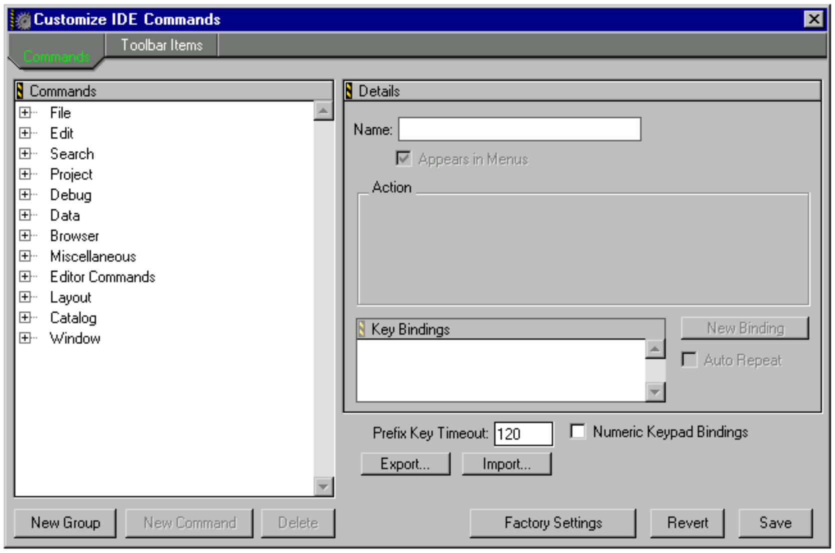


Figure 8-24 Customize IDE Commands window

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 8-25 on page 8-41).

   ——— **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 pop-up 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.

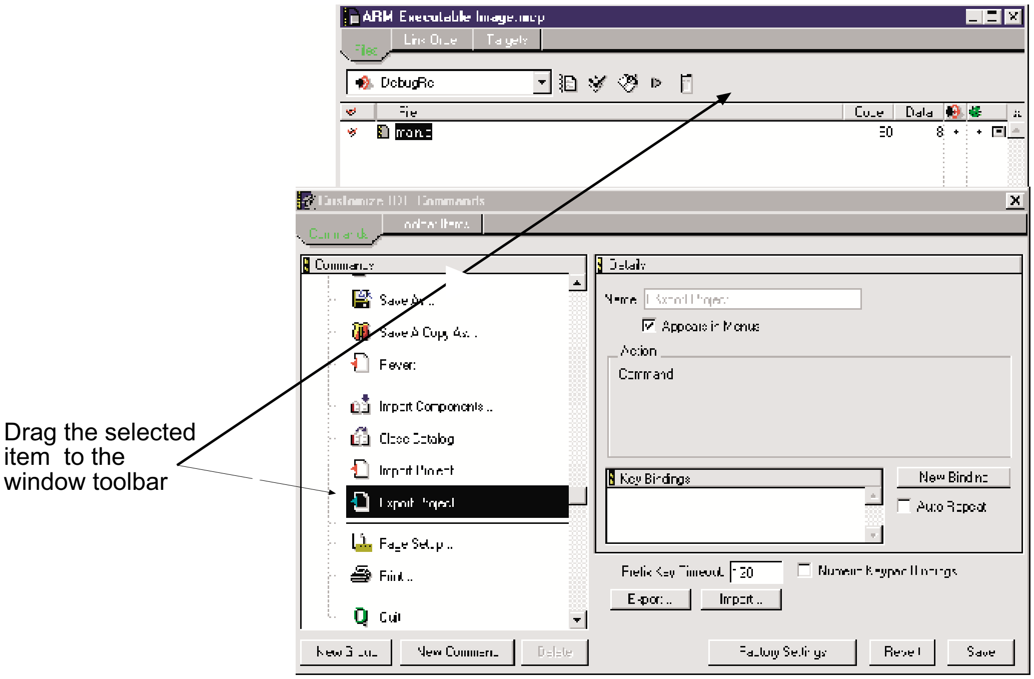


Figure 8-25 Dragging a toolbar element

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 pop-up 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 9
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:

9.1 About configuring a build target

The Target Settings window handles 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 ATPCS options for your project, you must configure the options for each of the:

- Debug build target
- DebugRel build target
- Release build target.

See *Using ARM-supplied project stationery* on page 2-24 for more information on default ARM stationery. See also *Setting the current build target* on page 2-55 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 9-2 on page 9-5 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 ARM assembler and compilers, including ATPCS 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 ARM linker options and fromELF options. You can use this panel to configure fromELF to convert ELF output images to a number of binary formats.

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.

9.1.1 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 ARM compilers
- the ARM linker, to output executable ELF format images
- the AXD 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 ATPCS panels for the compilers and assembler, to ensure that the settings are appropriate to your target hardware and procedure call standard preferences
- the ARM linker panels, to ensure that the appropriate output is produced
- the ARM fromELF panels, if you are using fromELF to produce a ROMable image or disassembled source.

9.1.2 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 2-35 for more information.

9.2 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 9-8
- *Configuring assembler and compiler language settings* on page 9-31
- *Configuring linker settings* on page 9-61
- *Configuring editor settings* on page 9-75
- *Configuring the debugger* on page 9-77.

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.

9.2.1 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 2-15 for more information.
2. Use the build target pop-up menu to select the build target you want to configure (Figure 9-1).

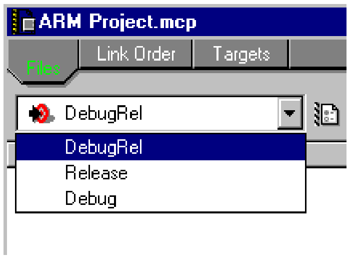


Figure 9-1 Select build target

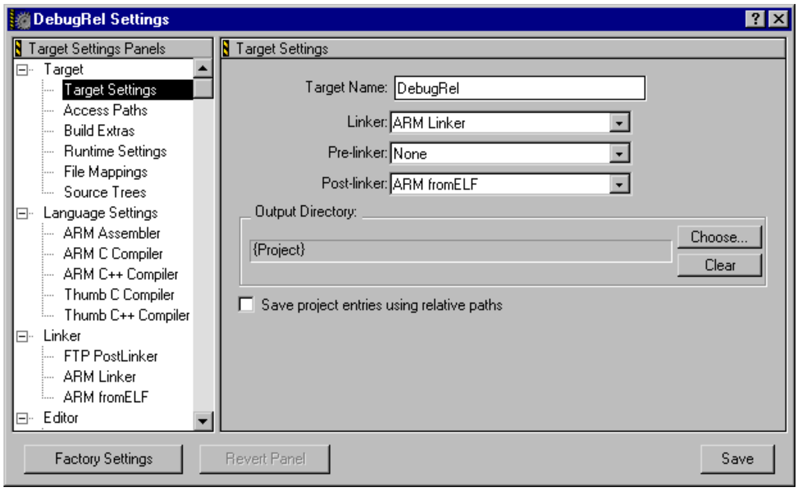
3. Either:
   - click the **Target Settings** button in the Project window.
   - select the **Settings** menu item for your build target from the **Edit** menu.

     The name of the **Settings** menu item matches the build target for your currently selected project. For example, if your current build target is DebugRel, the **Settings** menu item is named **DebugRel 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 9-2 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 9-8 for more information on specifying a linker.



Target Name: DebugRel
Linker: ARM Linker
Pre-linker: None
Post-linker: ARM fromELF
Output Directory:
{Project}
☐ Save project entries using relative paths

Figure 9-2 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.

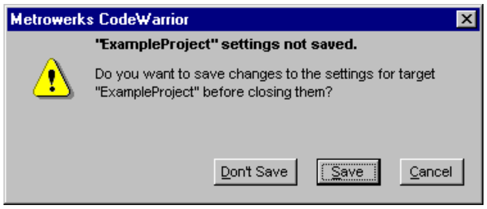
   Each panel contains related options that you can set. The options you select apply to the currently selected build target in the active project.

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 *Saving or discarding changes* on page 9-6 for more information on applying the changes you have made.

Saving 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 9-3).



"ExampleProject" settings not saved.

Do you want to save changes to the settings for target "ExampleProject" before closing them?

Figure 9-3 Settings Confirmation dialog box

Click one of:

- **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 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:

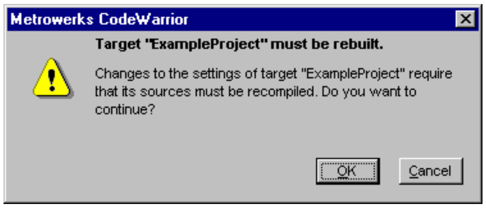
Factory Settings

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 Panel

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.

Save Click this button to commit any changes you have made in any of the panels. If you have changed an option that requires that the project be recompiled, the CodeWarrior IDE displays a confirmation dialog box (Figure 9-4 on page 9-7). Click **OK** or **Cancel** depending on whether you want to keep your changes or not.


Metrowerks CodeWarrior

Target "ExampleProject" must be rebuilt.

Changes to the settings of target "ExampleProject" require that its sources must be recompiled. Do you want to continue?

Figure 9-4 Rebuild confirmation dialog box

9.3 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* on page 9-8
- *Configuring access paths* on page 9-11
- *Configuring build extras* on page 9-20
- *Configuring runtime settings* on page 9-23
- *Configuring file mappings* on page 9-23
- *Configuring source trees* on page 9-25.

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.

9.3.1 Configuring target settings

The Target Settings panel enables you to specify basic settings for your project such as:

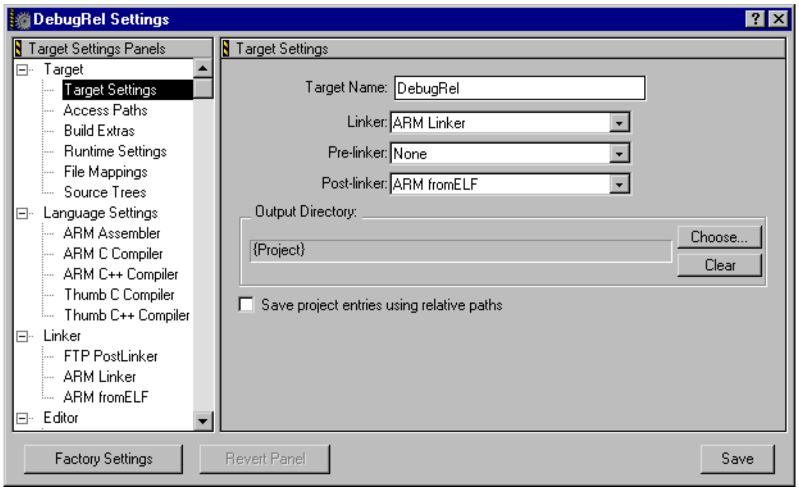
- the name of the target
- the linker and post-linker to use
- the output directory, and whether to use relative or absolute paths for files in the project.

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.

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 9-4).
2. Click **Target Settings** in the Target Settings Panels list to display the configuration panel (Figure 9-5).



Target Settings

Target Name: DebugRel
Linker: ARM Linker
Pre-linker: None
Post-linker: ARM fromELF
Output Directory: {Project}
☐ Save project entries using relative paths

Figure 9-5 Target Settings panel

3. Specify the settings for your target:

Target Name

Use the Target Name text field to set or change the name of your current build target. This is not the name of your final output file. See *Output file naming conventions and locations* on page 2-73 for more information on how CodeWarrior for the ARM Developer Suite names output files.

Linker Select a linker option from the pop-up menu. You can choose:

ARM linker

Use the ARM linker to link the output from the assembler and compiler. See *Configuring the ARM linker* on page 9-62 for more information on linker options.

ARM librarian

Use the ARM librarian to create a library from the compiler and linker output.

None Do not define a linker. This means that source files will not be compiled or assembled, because your choice of linker determines the compiler and assembler that CodeWarrior calls. You can use this option if you want to define a prelink or postlink step only, or if you want to use the CodeWarrior IDE to maintain a collection of non-source files.

Note

See *Configuring file mappings* on page 9-23 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 CodeWarrior for the ARM Developer Suite.

Post-Linker

Select a post-linker to process output from the linker. Choose one of:

None Do not use a post-linker.

ARM fromELF

Send the output from the linker to the ARM fromELF utility. FromELF processes the output using the options set in the fromELF configuration panel. Use this option to convert ELF images output by the linker to other formats. See *Configuring fromELF* on page 9-72 for more information.

FTP Post-Linker

This option is not used by CodeWarrior for the ARM Developer Suite.

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 2-84 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. You can define the output directory relative to a defined source tree. See *Configuring source trees* on page 9-25 for more information.

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 this setting is not selected, project entries are stored by name. See *Re-search for files* on page B-14 and *Reset project entry paths* on page B-14 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, CodeWarrior for the ARM Developer Suite does not allow you to add multiple copies of source files that produce output objects. See *Filename requirements* on page 2-38 for more information.

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

9.3.2 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 The ARM tools search these paths for:

- User header files. These are header files that you include with a `#include"…"` statement.
- User libraries. These are libraries that correspond to your `#include"…"` header files.
- 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 ARM tools search these paths for:

- C++ system header files. These are C++ header files included with a `#include <…>` statement. See *Default header file search paths* on page 9-12 for more information on how the ARM C and C++ compilers search for C system header files.

- System libraries. These are the corresponding libraries for the system header files.

By default, the system paths list contains:

- `{Compiler}lib`. The ARM C and C++ compilers use this path to locate the `armlib` directory containing the C standard libraries. The ARM C++ compilers use this path to locate the `cpplib` directory containing the standard C++ libraries.
- `{Compiler}include`. The default directory for the ARM standard C++ library header files.

Note

Previous releases of the ARM development tools used the `ARMLIB` environment variable to define the location of C standard libraries. You can configure the ARM linker to use the `ARMLIB` environment variable, rather than the System access paths defined here. See *Configuring linker options* on page 9-66 for more information.

Caution

If you use the `ARMLIB` environment variable with the CodeWarrior IDE, and you are using The ARM Developer Suite and the ARM Software Development Toolkit on the same machine, you must configure `ARMLIB` to point to the correct libraries for the development system you are using. The ADS installer gives you the option of overwriting the value of `ARMLIB` when you install. The ADS standard libraries are not compatible with the standard libraries for the SDT.

Default header file search paths

The default search path for header files are defined by the Access path options you have chosen. The ARM project stationery defines the following access paths:

User Paths `{Project}`. This is the directory in which the project file is located.

System Paths

`{Compiler}Include` and `{Compiler}Lib`, where `{Compiler}` is the directory in which the ARM Developer Suite is installed.

Note

- The ARM compilers use the search paths defined in the CodeWarrior Access Paths configuration panel when they are called from the CodeWarrior IDE. This means that the default search paths are different from those used by the compilers when they are invoked from the command line. In particular:
  - the `ARMINC` environment variable is ignored
  - the `:mem` directory is not searched
  - there is no *current place* (Berkeley search rules are not followed).

  If you are using a default ADS installation, these differences have no affect. If you have made changes to your default installation, such as changing the value of `ARMINC`, or editing system header files, the behavior of the command line tools and the CodeWarrior IDE will be different. See the compiler chapter of the *ADS Compiler, Linker, and Utilities Guide* for details of how the compilers search for header files when invoked from the command line.

- The CodeWarrior IDE stops searching when it finds the first match for a header file. If you have two header files with the same name in your project, the CodeWarrior IDE does not search the second header file.

- You can speed up compilation if you ensure that search paths defined in the CodeWarrior IDE are not searched recursively. See *Setting access path options* on page 9-14 for more information.

You can change the default search paths in the following ways:

- Select the **Always Search User Paths** option (see Figure 9-6 on page 9-14) to search for system header files in the same way as user header files.

- Enter `-I`, `-j`, `-fd`, and `-fk` command-line options in the Equivalent Command Line text box. See the compiler chapter in the *ADS Compiler, Linker, and Utilities Guide* for a description of these options. See *Using the Equivalent Command Line text box* on page 9-29 for more information on specifying command-line options in the CodeWarrior IDE.

If your standard header files or libraries are not in the default access search paths, the ARM tools cannot find them when compiling, linking, or running your project. See *Adding an access path* on page 9-16 for information on adding access paths.

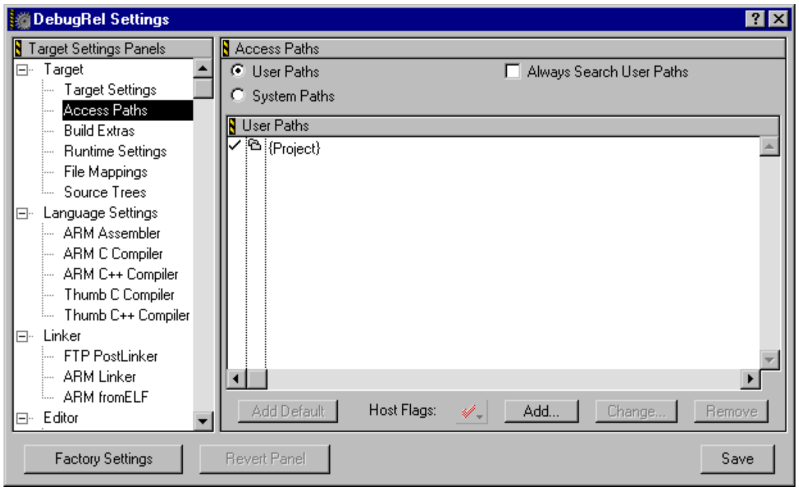
Setting access path options

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 9-4).
2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (Figure 9-6).



[✓ User Paths] System Paths
☐ Always Search User Paths
✓ {Project}

Figure 9-6 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:

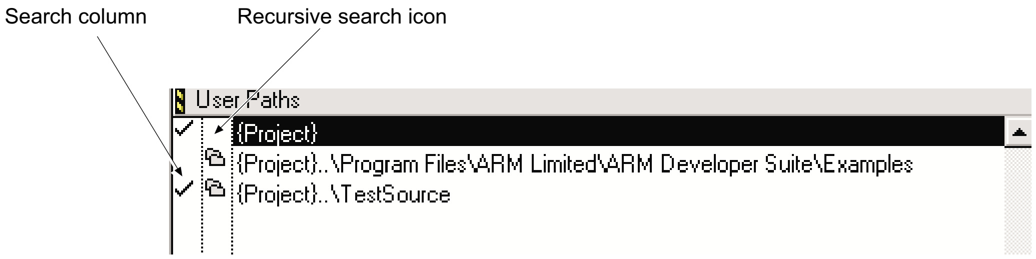
Specifying recursive searches

Note

It is strongly recommended that you do not use recursive searching in complex, or multi-user projects. See *Configuring CodeWarrior for complex or multi-user projects* on page 2-51 for more information.

Click in the Recursive Search column next to a folder name to toggle searching recursively through the folder (see Figure 9-7):

- 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.



{Project}
{Project}..\Program Files\ARM Limited\ARM Developer Suite\Examples
{Project}..\TestSource

Figure 9-7 Access paths detail

Note

You can speed up project compilation 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 9-7):

- if a check mark is displayed next to the name of a folder, the folder is searched from the current host computer.
- if a check mark is not displayed, the folder is not searched from the current host computer.

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.

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 9-12 for more information.

Note

It is strongly recommended that you select this option for complex, or multi-user projects. See *Configuring CodeWarrior for complex or multi-user projects* on page 2-51 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 **Save** to save 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 9-19 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 9-4).
2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (see Figure 9-6 on page 9-14).
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 9-8).

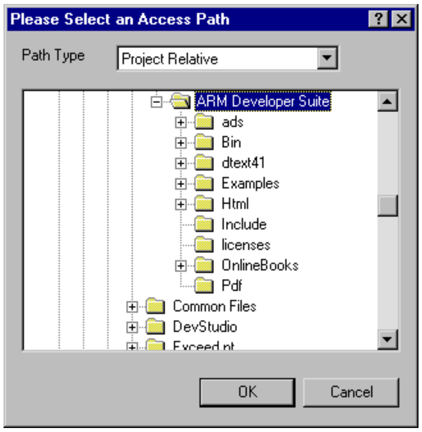


Figure 9-8 Select an Access Path dialog box

5. Select the path type you require from the **Path Type** pop-up menu.

Note

You can use relative paths to enable projects to contain two or more files with identical names. However, for large projects, using relative paths 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.

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 pop-up menu contains the name of any source trees you have defined. If you select a source tree, the CodeWarrior IDE defines the access path of the added folder relative to a folder defined in either a build target source tree, or a global source tree. See *Configuring source trees* on page 9-25 for more information on defining source trees for a specific build target. See *Configuring global source trees* on page 8-11 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 onto the Access Paths list pane. The path is added as:

- a project-relative path if the directory is on the same as volume 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 **Save** to save 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 9-4).
2. Click **Access Paths** in the Target Settings Panels list to display the configuration panel (see Figure 9-6 on page 9-14).
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 9-8 on page 9-17). Use the dialog box to navigate to the new folder location. See *Adding an access path* on page 9-16 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 **Save** to save your changes.

Adding the default access path

You can click the **Add Default** button to add the default CodeWarrior 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 paths for the project are:

- `{Compiler}include`
- `{Compiler}lib`

The **Add Default** button does not add these paths 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 9-16 for information.

Host Flags

The **Host Flags** pop-up menu 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.

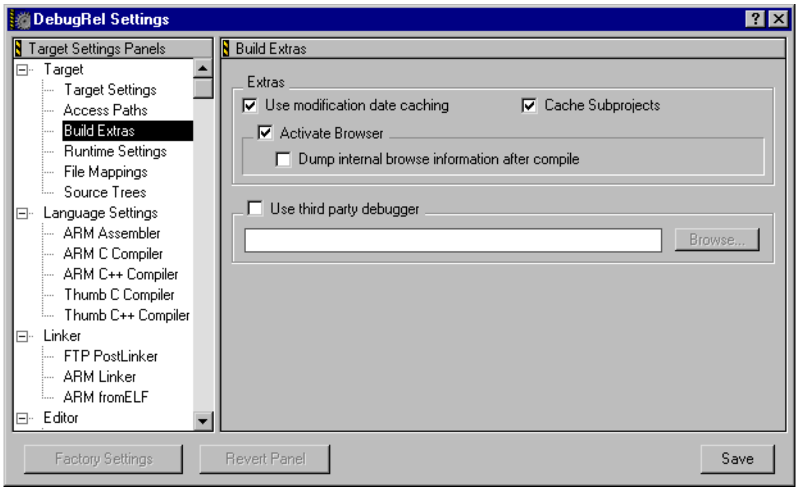
9.3.3 Configuring build extras

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 9-4).
2. Click **Build Extras** in the Target Settings Panels list to display the configuration panel (Figure 9-9).



☑ Use modification date caching
☑ Cache Subprojects
☑ Activate Browser
☐ Dump internal browse information after compile
☐ Use third party debugger

Figure 9-9 Build Extras settings panel

3. Select values for the following options:

Use modification date caching

Select this option to instruct he CodeWarrior IDE not to check the modification dates of files you have changed outside the CodeWarrior IDE. Select this option if you edit files with the CodeWarrior editor only, and are working in a single user environment. Selecting this option reduces compilation time.

Deselect this option if you have configured the CodeWarrior IDE to use a third-party editor, or you are working on a multi-user project with shared access to source files. See *Configuring IDE extras* on page 8-7 for more information on using a third-party editor. See *Configuring CodeWarrior for complex or multi-user projects* on page 2-51 for more information on using CodeWarrior in a complex build environment.

Cache Subprojects

Select this option to:

- Improve multiproject updating and linking.
- Enable the CodeWarrior browser to include browser information from target subprojects. See Chapter 7 *Browsing Source Code* for more information on the browser.

You can deselect this option to reduce the amount of memory required by the CodeWarrior IDE.

Activate Browser

Select this checkbox to generate the information needed by the CodeWarrior browser. The information is generated the next time you build your project. You cannot open browser windows for your project if this option is not selected.

See *Making a project* on page 2-77 for more information about rebuilding your project. For more information on browser settings and options, see Chapter 7 *Browsing Source Code*.

——— **Note** ———

The contextual menu features of the browser work in the CodeWarrior editor, in addition to all browser windows. You should consider enabling the browser, 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.

———————

If the **Activate Browser** option is selected you can also select the **Dump internal browser information after compile** option to view the raw browser information that a plug-in compiler or linker provides for the CodeWarrior IDE. This information is useful only if you are developing plug-ins for the CodeWarrior IDE.

——— **Note** ———

- You cannot build plug-ins for the CodeWarrior IDE with the ARM tool chain.
- Compile only single files, or small files with this option selected. The information that the CodeWarrior IDE displays can be very large if you compile an entire project.

———————

Use third party debugger

——— **Note** ———

Use the ARM Debugger panel to specify a third-party debugger in preference to this option. The ARM Debugger panel enables you to specify a command line for the third-party debugger. See *Choosing a debugger* on page 9-78 for details. If you select this option, it overrides the ARM Debugger panel option.

———————

You can select this checkbox to use any third-party debugger capable of loading and debugging ARM ELF/DWARF2 images, in place of the ARM debuggers. Enter the path to the debugger in the text field, or click the **Browse…** button to locate the debugger of your choice with the standard open file dialog.

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

9.3.4 Configuring runtime settings

The Configure runtime settings panel is not used by the ARM version of the CodeWarrior IDE.

9.3.5 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 C 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.

For example, if you create a project from the ARM Executable Image stationery, files with an extension of `.c` are mapped to the ARM C compiler. If you create a project from the Thumb Executable Image stationery, files with an extension of `.c` are mapped to the Thumb C compiler.

Note

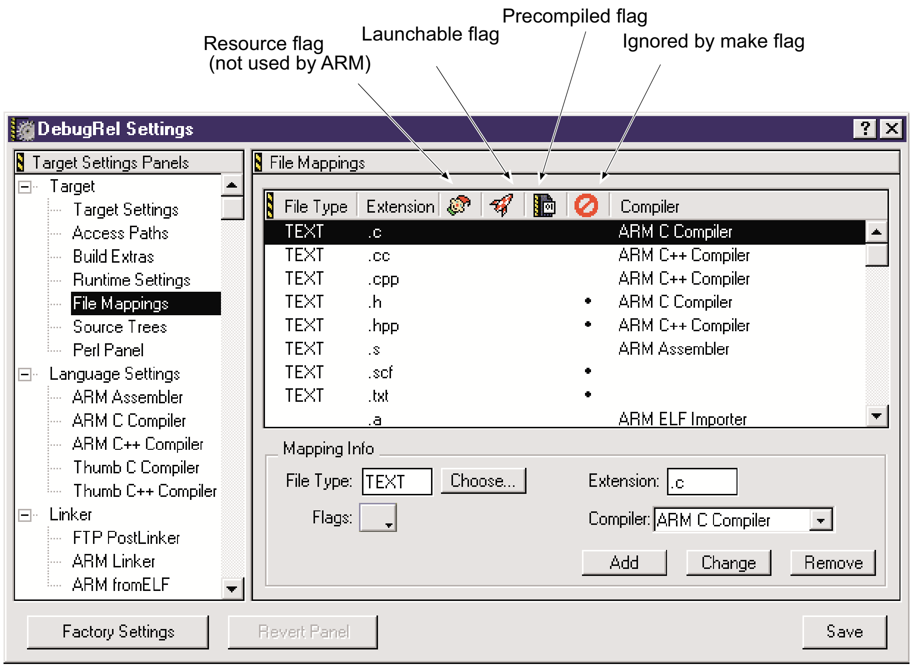
- File mappings determine whether the CodeWarrior IDE will recognize files in the project. If you have trouble adding files to your project, or if the CodeWarrior IDE refuses a folder or file that is dragged and dropped on the Project window, check the File Mappings settings panel to ensure that the 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 9-8 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 9-4).

2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel (Figure 9-10).



Target
- Target Settings
- Access Paths
- Build Extras
- Runtime Settings
- File Mappings
- Source Trees
- Perl Panel

Language Settings
- ARM Assembler
- ARM C Compiler
- ARM C++ Compiler
- Thumb C Compiler
- Thumb C++ Compiler

Linker
- FTP PostLinker
- ARM Linker
- ARM fromELF

File Type	Extension					Compiler
TEXT	.c					ARM C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				●	ARM C Compiler
TEXT	.hpp				●	ARM C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf				●	
TEXT	.txt				●	
	.a					ARM ELF Importer

File Type: TEXT
Flags:
Extension: .c
Compiler: ARM C Compiler

Figure 9-10 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 9-10 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 Thumb C or C++ compilers, as appropriate.

3. Click the entry you want to modify in the File Mappings list and enter the File Type for the file. Alternatively, click the **Browse…** 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 **Compiler** pop-up menu 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.

6. Click the **Flags** pop-up menu 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 9-10 on page 9-24). The flags are:

   Resource File

   This flag does not apply to the ARM 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 with the ARM tool chain. However, if you can use the standard Metrowerks CodeWarrior development environment to write your own plug-ins to work with CodeWarrior for the ARM Developer Suite.

   ————————

   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.

7. Click either:
   - the **Add** button, to add a new file mapping
   - the **Change** button, to change the configuration for a selected file mapping.

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

9.3.6 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.

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. See *Configuring global source trees* on page 8-11 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.

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 9-4).
2. Click **Source Trees** in the Target Settings Panels list to display the configuration panel (Figure 9-11 on page 9-26). The source trees panel displays a list of currently defined source paths.

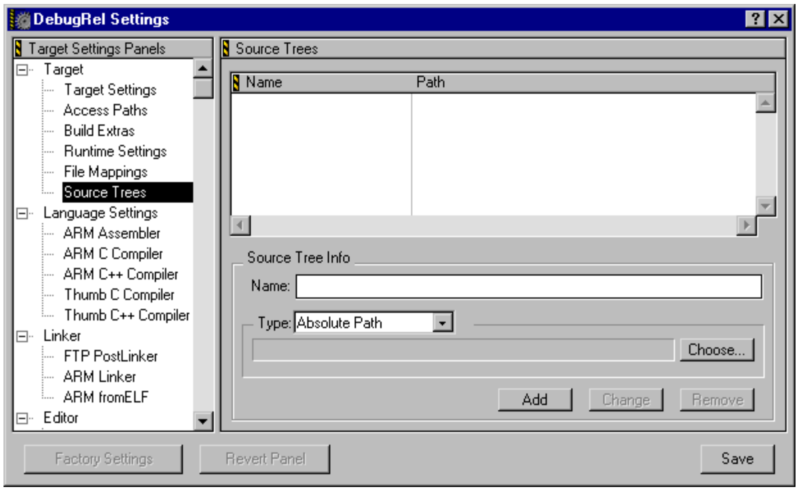


Figure 9-11 Source Trees panel

3. 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 and follow the steps below.
4. Click the **Type** pop-up menu 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 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. For example, to add the directory defined by the `ARMHOME` registry entry, enter:

     ```
     HKEY_LOCAL_MACHINE\SOFTWARE\ARM Limited\ARM Developer
     Suite\v1.1\ARMHOME
     ```
6. Click **Add** to add a new source tree, or click **Change** if you are modifying an existing source tree.
7. Click **Save** to save 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 (Figure 9-12). See *Configuring access paths* on page 9-11 for more information on adding access paths to CodeWarrior projects.

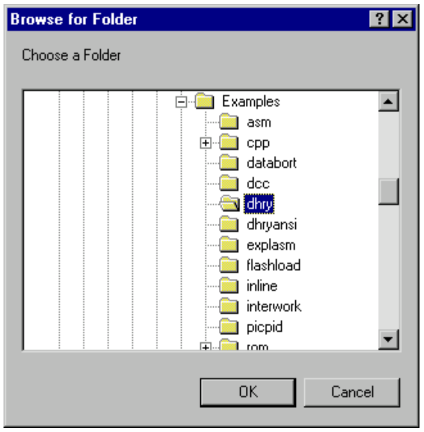


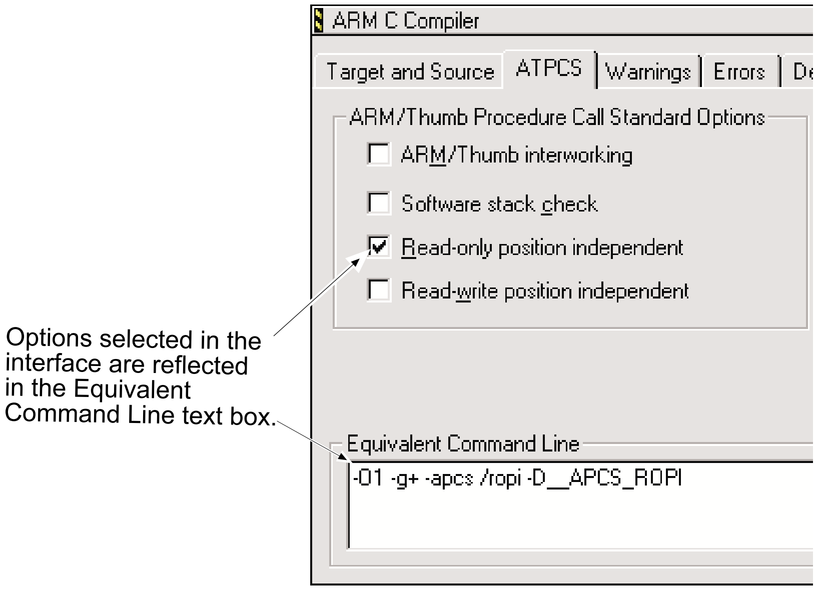
Figure 9-12 Example source path

9.4 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:

- ARM Language Settings panels
- ARM Linker panels
- ARM Debugger panels.

Figure 9-13 shows an example for the ARM C compiler configuration panel.


ARM C Compiler
Target and Source
ATPCS
Warnings
Errors
ARM/Thumb Procedure Call Standard Options
ARM/Thumb interworking
Software stack check
Read-only position independent
Read-write position independent
Options selected in the interface are reflected in the Equivalent Command Line text box.
Equivalent Command Line

```
-O1 -g+ -apcs /ropi -D__APCS_ROPI
```

Figure 9-13 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 **Read-only position independent** option in the Compiler ATPCS panel, the Equivalent Command Line text box changes to display `-apcs /ropi` 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 compilers, 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 *ADS Compiler, Linker, and Utilities Guide* for more information on the command-line options to the ARM tools.

- Enables you to copy and paste language settings between build targets.

Press the Enter key after you have edited the command line to apply the options and update the panel interface settings.

9.5 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 compilers and assembler:

Assembler Use this panel to configure options for the ARM assembler, including:

- the processor type or architecture version, and other capabilities of your target hardware
- ATPCS options
- debug and optimization options
- listing options, and other miscellaneous options.

In addition, you can use this panel to predefine variable values for the assembler. See *Configuring the ARM assembler* on page 9-32 for details.

Compilers There is a separate configuration panel for each of the ARM compilers in the language settings group. These panels are identical. Use the compiler configuration panels to configure options including:

- the processor type or architecture version, and other capabilities of your target hardware
- ATPCS options
- language mode, for example ANSI C or strict ANSI C
- debug and optimization options
- warning, error, and other preferences.

See *Configuring the compilers* on page 9-42 for details of options common to all the compilers.

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. 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 9-61
- *Configuring the debugger* on page 9-77.

9.5.1 Configuring the ARM assembler

This section describes how to configure the ARM assembler 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 assembler chapter of the *ADS Compiler, Linker, and Utilities Guide*.

——— **Note** ———

Many of the configuration settings described here are optional. However, you should review at least the Target options, ATPCS options, and Debug table generation options to ensure that they are suitable for your project.

————————————

The configuration options are described in:

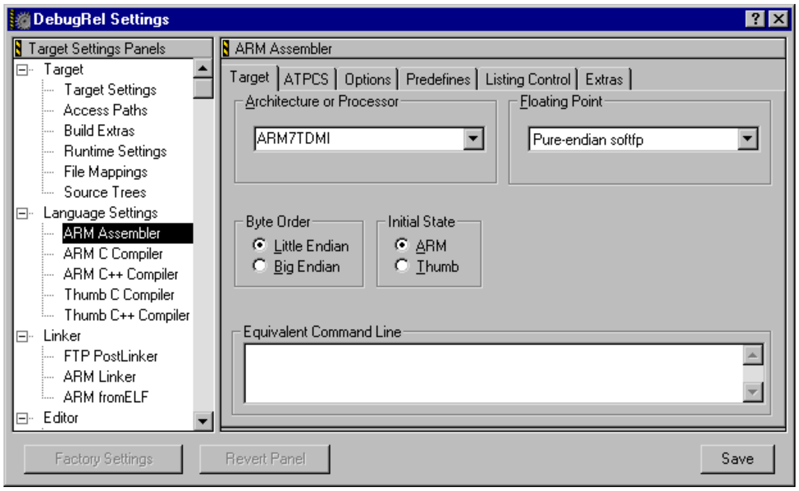
- *Configuring the target* on page 9-32
- *Configuring assembler ATPCS options* on page 9-34
- *Configuring assembler options* on page 9-37
- *Configuring predefined variables* on page 9-38
- *Configuring code listings* on page 9-40
- *Reading assembler options from a file* on page 9-41.

These sections give general descriptions of the available assembler options. Where necessary, you are referred to more detailed descriptions in the Assembler chapter of the *ADS Compiler, Linker, and Utilities Guide*.

Configuring the target

Use the Target panel to configure the target processor and architecture for the ARM assembler:

1. Display the Target Settings panel for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Click **ARM Assembler** in the Target Settings Panels list and click the **Target** tab to display the configuration panel (Figure 9-14 on page 9-33).



Architecture or Processor: ARM7TDMI

Floating Point: Pure-endian softfp

Byte Order: [✓ Little Endian] Big Endian

Initial State: [✓ ARM] Thumb

Equivalent Command Line

Figure 9-14 ARM Assembler Target panel

3. Select values for the following options:

Architecture or Processor

Select the processor or architecture for your target hardware from the pop-up menu. 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. If you specify this option it overrides any implicit FPU set by the `-cpu` option. Floating-point instructions produce either errors or warnings if assembled for the wrong target FPU.

The assembler sets a build attribute corresponding to *`name`* in the object file. The linker determines compatibility between object files, and selection of libraries, accordingly.

Valid values are:

FPA format and instructions

Selects hardware Floating Point Accelerator.

VFPv1 format and instructions

Selects hardware vector floating-point unit conforming to architecture VFPv1.

VFPv2 format and instructions

Selects hardware vector floating-point unit conforming to architecture VFPv2.

Old-style mixed-endian softfp

Selects software floating-point library with mixed-endian doubles.

Pure-endian softfp

Selects software floating-point library (FPLib) with pure-endian doubles. This is the default.

VFP with softvfp calling standard

Selects hardware Vector Floating Point unit.

To `armasm`, this is identical to `-fpu vfpv1`. See the *C and C++ Compilers* chapter in *ADS Compiler, Linker, and Utilities Guide* for details of the effect on software library selection at link time.

No floating point

Selects no floating-point option. This makes your assembled object file compatible with any other object file.

Byte Order

Select the byte order used by your target hardware.

Initial State

Select the state that the processor is expected to be in when your code is executed. This option is available only if you have selected an architecture or processor that supports the Thumb instruction set.

Note

This option does not generate code to switch the processor state. All ARM processors start in ARM state. You must ensure that the processor is in the state you expect when your code is run. See the *ADS Developer Guide* for information on switching between ARM state and Thumb state.

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

Configuring assembler ATPCS options

Selecting ATPCS 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.

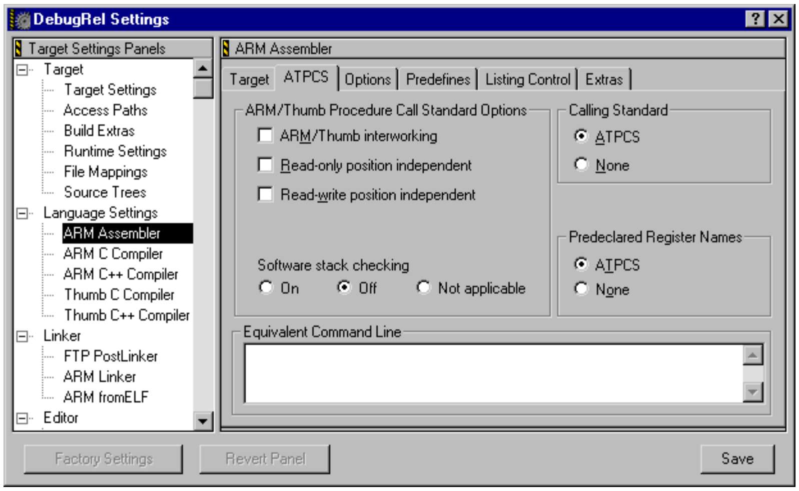
Note

Selecting ATPCS options does *not* provide checks that your assembly language code conforms to a given ATPCS variant. You must ensure that your assembly language code follows the conventions required by the ATPCS options you select. See the chapter on using the ATPCS in the *ADS Developer Guide* for more information.

If you expect to call your assembly language code from C or C++, you must ensure that your C and C++ compiler options are configured to use the same calling standard options. See *Configuring compiler ATPCS options* on page 9-46 for details.

To configure the ARM/Thumb Procedure Call Standard settings for the ARM assembler:

1. Display the Target Settings panel for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Click **ARM Assembler** in the Target Settings Panels list and click the **ATPCS** tab to display the configuration panel (Figure 9-15).



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

Software stack checking: On [✓ Off] Not applicable

Calling Standard: [✓ ATPCS] None

Predeclared Register Names: [✓ ATPCS] None

Equivalent Command Line

Figure 9-15 ARM Assembler ATPCS panel

3. Select values for the calling standard and predefined register names options:

Calling standard

Select **ATPCS** if your assembly language code is expected to conform to one of the ATPCS variants. You should select **ATPCS** 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 ATPCS calling conventions.

You can select **None** if you are writing standalone assembly language routines that are not called from other routines that expect ATPCS calling conventions to be followed. In this case you are responsible for maintaining your own register usage conventions.

Predeclared Register Names

Select ATPCS if you are writing assembly language routines that conform to one of the ATPCS variants and you want the assembler to recognize the conventional predefined register names, such as a1-a4. See the Assembler chapter of the *ADS Compiler, Linker, and Utilities Guide* for a complete list of predefined register names.

4. Select the ATPCS variant options that you require.

   The following options are available:

   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 when the assembler output is linked. See the description of Interworking in the *ADS 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 **Save** to save your settings.

Configuring assembler options

To configure other options, such as code checks, warnings, and debug options, for the ARM assembler:

1. Display the Target Settings panel for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Click **ARM Assembler** in the Target Settings Panels list and click the **Options** tab to display the configuration panel (Figure 9-16).

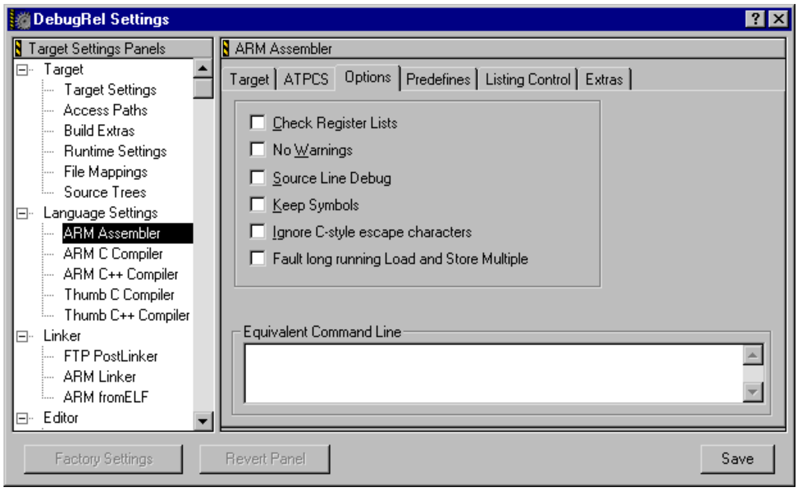


Figure 9-16 ARM Assembler Options panel

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. By default, when you select this option the **Keep Symbols** option is also selected.

   Keep Symbols

   Select this option to keep local labels in the symbol table of the object file, for use by the debugger.

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 Multiples

Select this option to instruct the assembler to fault `LDM` and `STM` instructions if the maximum number of registers transferred exceeds:

- five, for all `STM`s, and for `LDM`s that do not load the PC
- four, for `LDM`s that load the PC.

Avoiding large multiple register transfers can reduce interrupt latency on ARM systems that:

- do not have a cache or a write buffer (for example, a cacheless ARM7TDMI)
- use zero wait-state, 32-bit memory.

Note

Avoiding large multiple register transfers increases code size and decreases performance slightly.

Avoiding large multiple register transfers has no significant benefit for cached systems or processors with a write buffer.

Avoiding large multiple register transfers also has no benefit for systems without zero wait-state memory, or for systems with slow peripheral devices. Interrupt latency in such systems is determined by the number of cycles required for the slowest memory or peripheral access. This is typically much greater than the latency introduced by multiple register transfers.

4. Click **Save** to save 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 Assembler chapter in the *ADS Compiler, Linker, and Utilities 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 9-4).
2. Click **ARM Assembler** in the Target Settings Panels list and click the **Predefines** tab to display the configuration panel (Figure 9-17 on page 9-39).

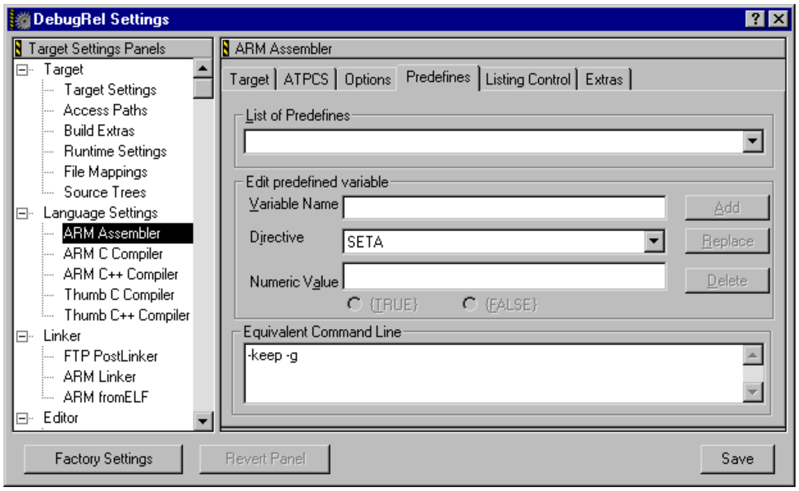


Figure 9-17 ARM Assembler Predefines panel

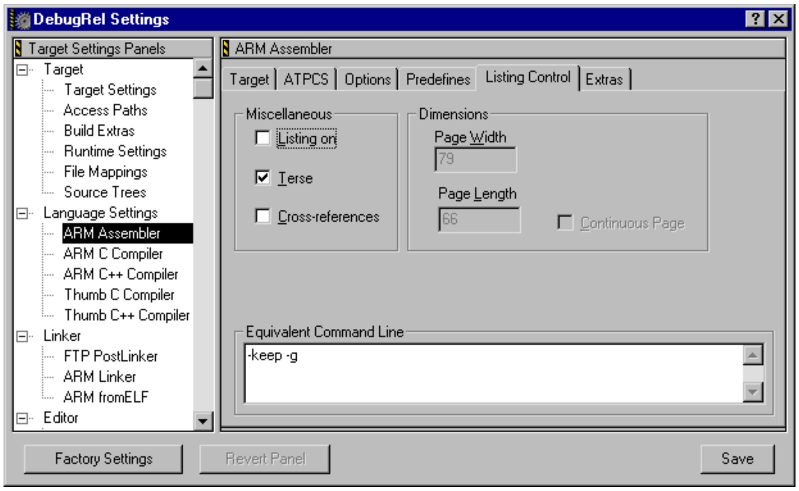
3. Enter the name of the variable in the Variable Name field. Alternatively, if you want to modify or delete an existing predefined variable, select the variable from the List of Predefines pop-up menu.
4. Select the directive you require. Use:
   - `SETA`, for numeric values
   - `SETS`, for string 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.
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 **Save** to save 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 9-4).

2. Click **ARM Assembler** in the Target Settings Panels list and click the **Listing Control** tab to display the configuration panel (Figure 9-18).



Miscellaneous

- [ ] Listing on
- [x] Terse
- [ ] Cross-references

Dimensions

Page Width: 79

Page Length: 66

- [ ] Continuous Page

Equivalent Command Line: -keep -g

Figure 9-18 ARM 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.

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.

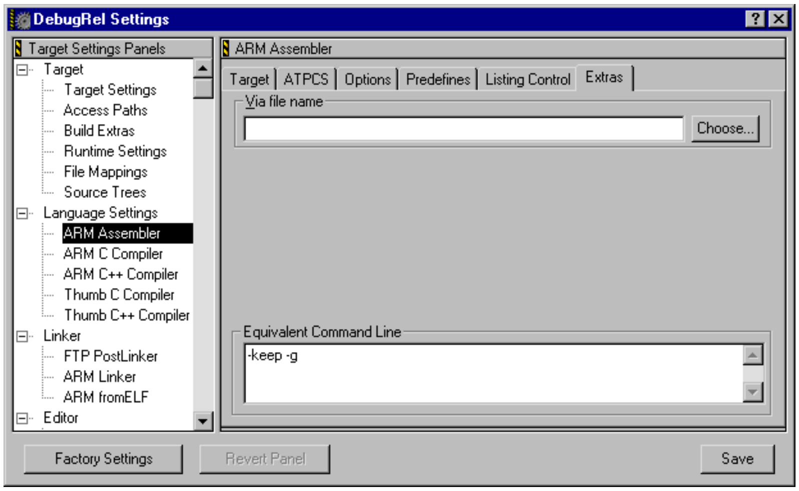
4. Click **Save** to save your settings.

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. See the *ADS Compiler, 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 9-4).
2. Click **ARM Assembler** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 9-19).



Equivalent Command Line: -keep -g

Figure 9-19 ARM 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 options are processed after the options specified in the configuration panels, and override them if there is a conflict.
4. Click **Save** to save your settings.

9.5.2 Configuring the compilers

This section describes how to configure compiler options that are common to each of the ARM compilers:

- the ARM C compiler, armcc
- the Thumb C compiler, tcc
- the ARM C++ compiler, armcpp
- the Thumb C++ compiler, tcpp.

——— **Note** ———

Many of the configuration settings described here are optional. However, you should review at least the Target options, ATPCS options, and Debug table generation options to ensure that they are suitable for your project.

————————

Each compiler has its own configuration panel consisting of a number of tabbed panes. The panels are listed in the Target Settings Panels list (see Figure 9-20 on page 9-43 for an example) when you select the ARM linker or the ARM librarian as the linker in the Target Settings panel. See *Configuring target settings* on page 9-8 for more information on selecting the linker in the Target Settings panel.

The configuration options are described in:

- *Configuring the target and source* on page 9-42
- *Configuring compiler ATPCS options* on page 9-46
- *Configuring warnings* on page 9-48
- *Configuring errors* on page 9-52
- *Configuring debug and optimization* on page 9-54
- *Configuring the preprocessor* on page 9-55
- *Configuring code generation* on page 9-56.

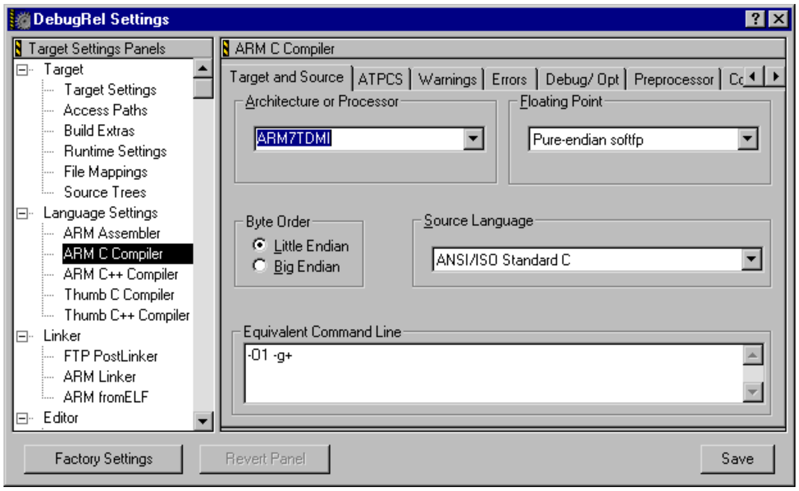
These sections give general descriptions of the available compiler options. Where necessary, you are referred to more detailed descriptions in the C and C++ Compiler chapters of the *ADS Compiler, Linker, and Utilities Guide*.

Configuring the target and source

To configure the target processor and architecture for the ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Target and Source** tab to display the configuration panel (Figure 9-20).



Architecture or Processor: ARM7TDMI
Floating Point: Pure-endian softfp
Byte Order: [✓ Little Endian] Big Endian
Source Language: ANSI/ISO Standard C
Equivalent Command Line: -01 -g+

Figure 9-20 ARM 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 pop-up menu. The pop-up menu 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 ARM compilers **ARM10200E** implies **VFPv2 format and instruction**. The implied option is overridden if you specify an explicit floating point option. If no processor or floating-point options are specified, **Pure-endian softfp** is used.

Floating Point

Select the floating-point system you are using. Valid values are:

FPA format and instructions

Select this option if you are targeting a system with a hardware Floating Point Architecture unit. This option is not available for the Thumb compilers.

VFPv1 format and instructions

Select this option to target hardware vector floating-point unit conforming to architecture VFPv1, such as the ARM10™ rev 0. This option is not available for the Thumb compilers.

VFPv2 format and instructions

Select this option to target hardware vector floating-point unit conforming to architecture VFPv2, such as the ARM10200E. This option is not available for the Thumb compilers.

Old-style mixed-endian softfp

Select this option to use the software floating-point library and your source code uses FPA-format double-precision floating-point representations.

Pure-endian softfp

Select this option to use the software floating-point library with pure-endian doubles. This option specifies that your code uses VFP format double-precision floating-point representations, but does not use any floating-point coprocessor instructions. This is the default.

VFP with softvfp calling standard

Selects software floating-point library with pure-endian doubles, software floating-point linkage, and requiring a VFP unit. Select this option if you are interworking Thumb code with ARM code on a system that implements a VFP unit.

If you select this option the compiler behaves exactly as for **Pure-endian softfp** except that it links with VFP-optimized floating-point libraries.

Note

If this option is specified for both armcc and tcc, it ensures that your interworking floating-point code is compiled to use software floating-point linkage. If you select **VFPv1 format and instructions**, or **VFPv2 format and instructions** for armcc you must use the `__softfp` keyword to ensure that your interworking ARM code is compiled to use software floating-point linkage. See the description of `__softfp` in the *ADS Compiler, Linker, and Utilities Guide* for more information.

No floating point

Select this option if you are neither targeting a hardware floating-point unit, nor the software floating-point library.

Byte Order

Select the byte order used by your target hardware.

Source language

Select the source language you require from the pop-up menu. See the compiler chapters of the *ADS Compiler, Linker, and Utilities Guide* for detailed information on C and C++ standards conformance and implementation details. Valid options for both C and C++ compilers are:

ANSI/ISO Standard C

Select this option to compile fairly strict ANSI standard C, with some characteristics removed, and some minor extensions, such as accepting C++ style comments (`//`), and accepting `$` characters in identifiers.

Strict ANSI/ISO Standard C

Select this option to enforce stricter adherence to the ANSI standard.

The ARM C++ compilers provide the following additional options:

ISO/IEC Standard C++

Select this option to compile standard ISO/IEC C++ with minor extensions.

Strict ISO/IEC Standard C++

Select this option to enforce stricter adherence to the ISO/IEC standard. For example, the following code gives an error when this option is selected, and a warning when compiled with the standard ISO/IEC option:

```
static struct T {int i;};
```

The static declaration is spurious because no object is declared. In strict C++ it is illegal.

Embedded C++

Select this option to compile C++ that closely conforms to the industry standard for embedded C++. Embedded C++ is a subset of standard C++ that is designed to encourage efficient code for use in embedded systems. The Embedded C++ standard is evolving. The proposed definition can be found on the web at `http://www.caravan.net/ec2plus`.

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

Configuring compiler ATPCS options

Selecting ATPCS options instructs the compiler to generate code that conforms to the appropriate ATPCS 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.

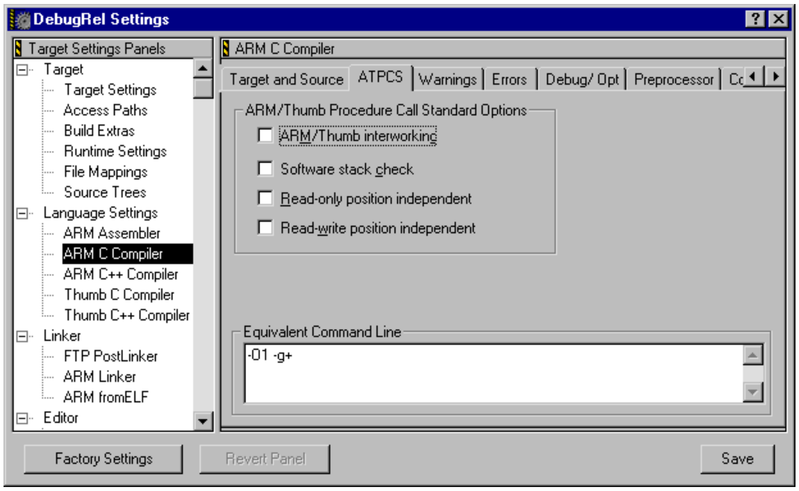
Note

Routines that are entered from exception vectors do not necessarily need the same settings as the rest of your project code. For example, software stack checking should be avoided in FIQ handlers in order to maximize the speed of the handler.

In addition, if you are calling routines written in ARM assembly language you must ensure that your assembly language code conforms to the appropriate ATPCS variant, and that the assembler is configured with the same ATPCS options. See *Configuring assembler ATPCS options* on page 9-34 for details. See also the ATPCS chapter in the *ADS Developer Guide* for more information.

To configure the ARM and Thumb procedure call standard options for the ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **ATPCS** tab to display the configuration panel (Figure 9-21).



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

Equivalent Command Line
```
-O1 -g+
```

Figure 9-21 ARM compiler ATPCS panel

3. Select the ATPCS 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. Only functions called from the other state need to be compiled for interworking. The linker generates suitable interworking veneers when the compiler output is linked. See the description of Interworking in the *ADS Developer Guide* for more information.

Software stack check

Select this option if you are writing code that performs stack checking in software.

Read-only position independent

Select this option if you are compiling code that you want to be position-independent. If this option is selected, the compiler:

- addresses read-only entities (code and data) pc-relative
- sets the PI attribute on read-only output sections.

Note

The ARM tools cannot determine if the final output image will be Read-Only Position-Independent until the linker finishes processing input sections. This means that the linker might emit ROPI error messages, even though you have selected this option.

Read-write position independent

Select this option to ensure that output data sections in your compiled code are addressed position-independently. If this option is selected, the compiler:

- Addresses writable data using offsets from the static base (sb). This means that:
  - data addresses can be fixed at run time
  - data can be multiply instanced
  - data can be, but does not have to be, position-independent.
- Sets the PI attribute on read-write output sections.

Note

The compiler does not force your writable data to be position-independent. This means that the linker might emit RWPI messages, even though you have selected this option.

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

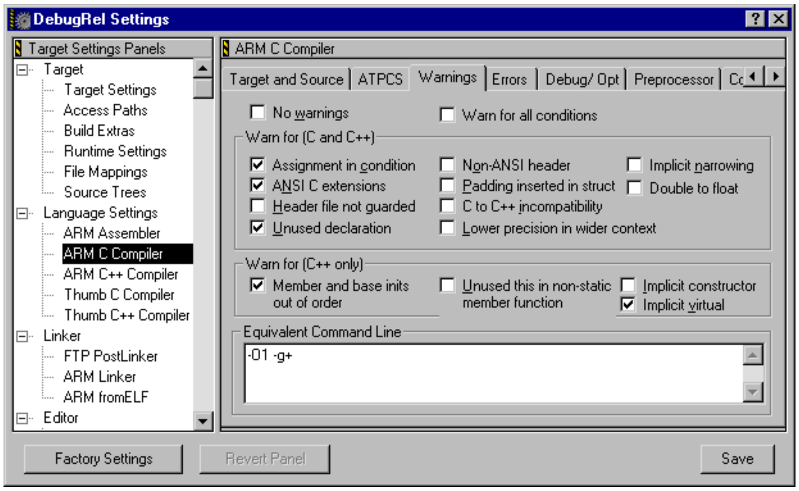
Configuring warnings

The compiler issues warning messages to warn about portability problems or other potential problems in your code. You can use the **Warnings** tab to configure the C and C++ compilers to suppress or enable specific warnings.

To configure warnings given by the ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Warnings** tab to display the configuration panel (Figure 9-22).



DebugRel Settings

Target Settings Panels
- Target
  - Target Settings
  - Access Paths
  - Build Extras
  - Runtime Settings
  - File Mappings
  - Source Trees
- Language Settings
  - ARM Assembler
  - ARM C Compiler
  - ARM C++ Compiler
  - Thumb C Compiler
  - Thumb C++ Compiler
- Linker
  - FTP PostLinker
  - ARM Linker
  - ARM fromELF
- Editor

ARM C Compiler

Target and Source | ATPCS | Warnings | Errors | Debug/ Opt | Preprocessor | Cc

- [ ] No warnings
- [ ] Warn for all conditions

Warn for (C and C++)
- [✓] Assignment in condition
- [✓] ANSI C extensions
- [ ] Header file not guarded
- [✓] Unused declaration
- [ ] Non-ANSI header
- [ ] Padding inserted in struct
- [ ] C to C++ incompatibility
- [ ] Lower precision in wider context
- [ ] Implicit narrowing
- [ ] Double to float

Warn for (C++ only)
- [✓] Member and base inits out of order
- [ ] Unused this in non-static member function
- [ ] Implicit constructor
- [✓] Implicit virtual

Equivalent Command Line

```
-O1 -g+
```

Factory Settings | Revert Panel | Save

Figure 9-22 ARM compiler Warnings panel

3. Select warning options that apply to both C and C++, as required:

   No warnings

   Select this option to turn off all warnings.

   Warn for all conditions

   Select this option to turn on all warnings, including those that are disabled by default.

   Assignment in condition

   Select this option to enable the warning:

   ```
   C2961W: use of "=" in condition context
   ```

This warning is issued when the compiler finds a statement such as:

```
if (a = b) {...
```

where it is possible that one of the following is intended:

```
if ((a = b) != 0) {...
if (a == b) {...
```

ANSI C extensions

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()`.

Header file not guarded

Select this option to enable the warning given when an unguarded header file is `#included`. An unguarded header file is one that is not wrapped in a declaration such as:

```
#ifndef foo_h
#define foo_h
/* body of include file */
#endif
```

Unused declaration

Select this option to enable *not used* warnings such as:

```
C2870W: variable 'y' declared but not used.
```

Warning are given for:

- local (within a function) declarations of variables, typedefs, and functions
- labels (always within a function)
- top-level static functions and static variables.

Non-ANSI header

Select this option to enable the warning message:

```
C2812W: Non-ANSI #include <…>
```

The ANSI C standard requires that you use `#include <…>` for ANSI C headers only. However, it is useful to disable this warning when compiling code that does not conform to this aspect of the standard. These warnings are suppressed by default unless you have selected a strict source language option in the Target and Source panel (see *Configuring the target and source* on page 9-42).

Padding inserted in struct

Select this option to enable warnings given when the compiler inserts padding in a **`struct`**.

For example:

```
C2221W: padding inserted in struct 's'
```

C to C++ incompatibility

For C code, select this option to enable warnings about future compatibility with C++. Warnings are suppressed by default. For example:

```
int *new(void *p) {  return p; }
```

results in the following warnings:

```
C2204W: C++ keyword used as identifier: 'new'
C2920W: implicit cast from (void *), C++ forbids
```

Lower precision in wider context

Select this option to enable the warning message:

```
Lower precision in wider context
```

that is given when code like the following is found:

```
long x; int y, z; x = y*z
```

where the multiplication yields an **`int`** result that is then widened to **`long`**. This warning indicates a potential problem when either the destination is **`long long`**, or when the target system defines 16-bit integers or 64-bit longs.

Implicit narrowing

Select this option to enable the `Implicit narrowing cast` warning message.

This warning is issued when the compiler detects the implicit narrowing of a long expression in an **`int`** or **`char`** context, or the implicit narrowing of a floating-point expression in an integer or narrower floating-point context.

Implicit narrowing casts such as these almost always cause problems when you move code that has been developed on a fully 32-bit system to a system in which integers occupy 16 bits and longs occupy 32 bits.

Double to float

Select this option to disable the warning:

```
C2621W: double constant automatically converted to float
```

This warning is given when the default type of unqualified floating-point constants is changed by the **Narrow double constants to float constants** option. This warning is switched on by default.

4. Select warning options specific to C++, if required:

   Member and base inits out of order

   Select this option to enable warnings given when base and member initializations are out of order. For example:

   ```
   struct T { T(int); int i, j; };T::T(int x) : j(x), i(j*3) { }
   ```

   The base and member initializations are done in declaration order (virtual bases are done first) no matter what the order specified in the definition of the constructor. In this case `i` is initialized before `j` and so `j*3` gives an undefined value.

   Unused this in non-static member function

   Select this option to enable the unused **`this`** warning. This warning is issued when the implicit **`this`** argument is not used in a non-static member function. It is applicable to C++ only. The warning can also be avoided by making the member function a static member function. The default is off. For example:

   ```
   struct T {
          int f() { return 42; }
   };
   ```

   results in the following warning:

   ```
   C2924W: 'this' unused in non-static member function
   ```

   To avoid the warning, use `static int f() ...`

   Implicit virtual

   Select this option to enable the implicit virtual warning that is given when a non-virtual member function of a derived class hides a virtual member of a parent class. For example:

   ```
   struct Base { virtual void f(); };
   struct Derived : Base { void f(); };
                       // warning 'implicit virtual'
   C2997W: 'Derived::f()' inherits implicit virtual from
   'Base::f()'
   ```

   Adding the **`virtual`** keyword in the derived class prevents the warning.

   Implicit constructor

   Select this option to enable the implicit constructor warning that is given when the code requires a constructor to be invoked implicitly. For example:

   ```
   struct X { X(int); };
   X x = 10;       // actually means, X x = X(10);
                   // See the Annotated C++
                   // Reference Manual p.272
   ```

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

Configuring errors

The compiler issues error messages to indicate serious problems in the code it is attempting to compile.

The options described below enable you to:

- turn specific recoverable errors off
- downgrade specific errors to warnings.

——— Caution ———

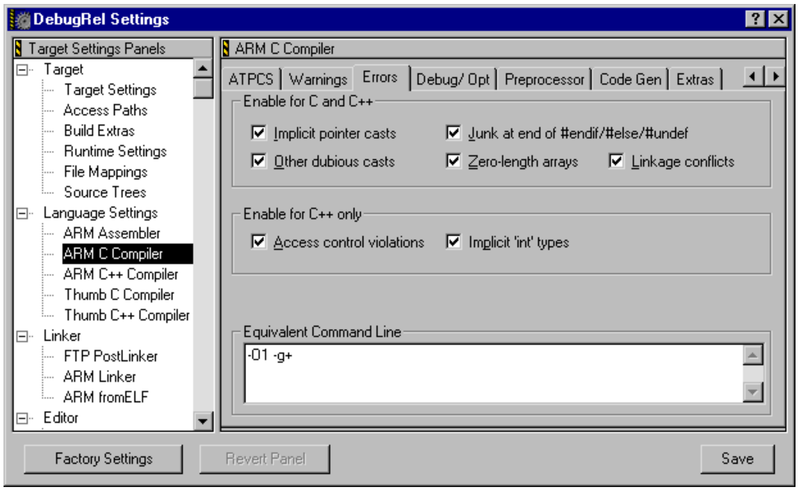
These options force the compiler to accept C and C++ source that normally produces errors. If you use any of these options to ignore error messages, it means that your source code does not conform to the appropriate C or C++ standard.

These options might be useful when importing code from other environments. However, they might allow the compiler to produce code that does not function correctly. It is generally better to correct the code than to use options to switch off error messages.

———————

To configure error messages issued by the ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Errors** tab to display the configuration panel (Figure 9-23).



Enable for C and C++
- ☑ Implicit pointer casts
- ☑ Junk at end of #endif/#else/#undef
- ☑ Other dubious casts
- ☑ Zero-length arrays
- ☑ Linkage conflicts

Enable for C++ only
- ☑ Access control violations
- ☑ Implicit 'int' types

Equivalent Command Line

```
-O1 -g+
```

Figure 9-23 ARM compiler Errors panel

3. Select error options that apply to both C and C++, if required:

   Note

   By default, all the following options are selected. Deselect an option to instruct the compiler to suppress or downgrade the error message.

   Implicit pointer casts

   Deselect this option to suppress all implicit cast error messages, such as the error message generated by casting a nonzero `int` to `pointer`. No warning message is given.

   Other dubious casts

   Deselect this option to downgrade error messages for illegal casts, such as those generated by casting `pointer` to `short`, to warnings.

   Junk at end of #endif/#else/#undef

   Deselect this option to suppress error messages generated as the result of extra characters at the end of a preprocessor line. No warning message is given.

   Zero-length arrays

   Deselect this option to suppress error messages arising from zero-length arrays. No warning message is given.

   Linkage conflicts

   Deselect this option to suppress error messages about linkage disagreements where functions are implicitly declared as `extern` and then later redefined as `static`. No warning message is given.

4. Select error options that apply only to C++, as required:

   Access control violations

   Deselect this option to downgrade access control errors to warnings. For example:

   ```
   class A { void f() {}; };   // private member
   A a;
   void g() { a.f(); }         // erroneous access
   ```

   Implicit int types

   Deselect this option to downgrade error messages produced by constructs such as:

   ```
   const i;
   //missing type specification for 'i' - 'int' assumed
   ```

   to warnings.

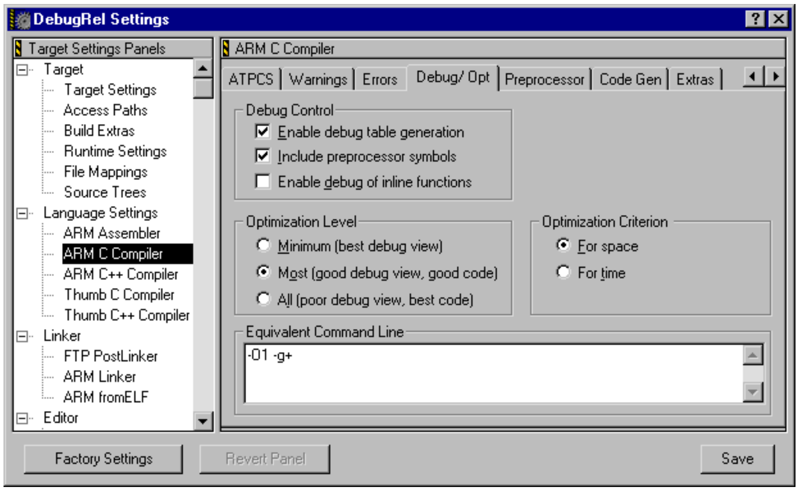
5. Click **Save** to save your changes.

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 compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Debug/Opt** tab to display the configuration panel (Figure 9-24 on page 9-54).



Debug Control
- [✓] Enable debug table generation
- [✓] Include preprocessor symbols
- [ ] Enable debug of inline functions

Optimization Level
- ○ Minimum (best debug view)
- ● Most (good debug view, good code)
- ○ All (poor debug view, best code)

Optimization Criterion
- ● For space
- ○ For time

Equivalent Command Line

```
-01 -g+
```

Figure 9-24 ARM compiler Debug and optimization panel

3. Select Debug control options:

Enable debug table generation

Select this option to instruct the compiler to generate DWARF2 debug tables. This option enables you to debug your output images at the source level. If this option is not selected, only limited debugging is available.

Include preprocessor symbols

Select this option to include preprocessor symbols in the compiler output.

Enable debug of inline functions	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:

None	Select this option to disable all compiler optimizations. Use this option in combination with enabled debug table generation to generate the best possible debug view of your output image.
Most	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.
All	Select this option to enable all compiler optimizations. This option results in a poor debug view of your output image due to code movement and register re-use. You can use this option with debug table generation turned off, or by linking with debug data discarded, to generate the most efficient code possible, after you have finished debugging.

5. Select the optimization criterion:

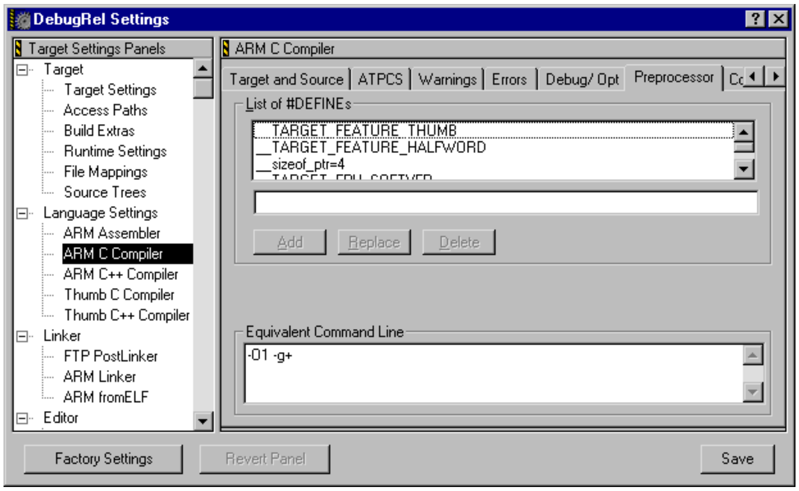
For space	Select this option to favor small code size over execution speed. This is the default.
For time	Select this option to favor execution speed over code size.

6. Click **Save** to save 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 ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Preprocessor** tab to display the configuration panel (Figure 9-25). The panel displays a list of all predefined macros.



```
__TARGET_FEATURE_THUMB
__TARGET_FEATURE_HALFWORD
__sizeof_ptr=4
```

```
-O1 -g+
```

Figure 9-25 ARM compiler Preprocessor panel

3. Double-click on a preprocessor macro definition in the List of #DEFINEs field to modify the definition. If you want to add a new definition, double-click on any existing definition. The name of the macro is displayed in the text field below the list.

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.

   Caution

   Do not replace the macro definitions predefined by ARM.

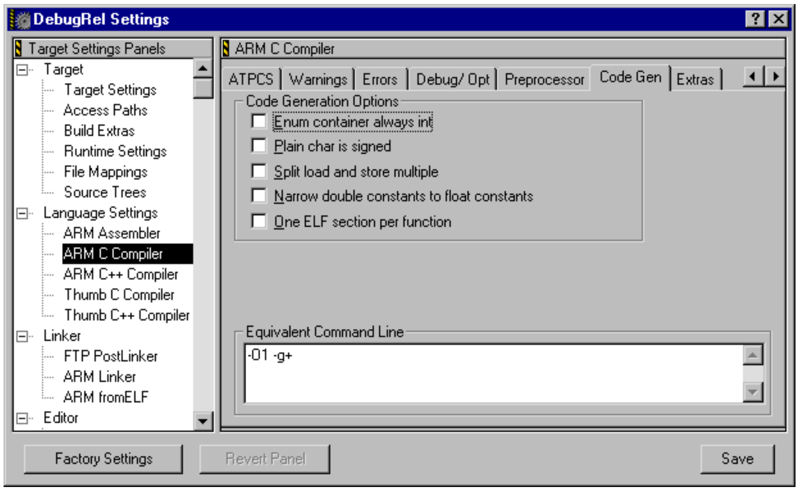
6. Click **Save** to save your changes.

Configuring code generation

To configure code generation options for the ARM compilers:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Depending on the compiler you are using, click the appropriate entry in the Target Settings Panels list and click the **Code Gen** tab to display the configuration panel (Figure 9-26 on page 9-57).



Code Generation Options

- [ ] Enum container always int
- [ ] Plain char is signed
- [ ] Split load and store multiple
- [ ] Narrow double constants to float constants
- [ ] One ELF section per function

Equivalent Command Line

```
-01 -g+
```

Figure 9-26 ARM 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 ANSI-compatible source. If used incorrectly, this option can cause errors in the resulting image.

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 ANSI-compatible source. If used incorrectly, this option can cause errors in the resulting image.

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.

This option can reduce interrupt latency on ARM systems that:

- do not have a cache or a write buffer (for example, a cacheless ARM7TDMI)
- use zero-wait-state, 32-bit memory.

Note

Using this option increases code size and decreases performance slightly.

This option does not split ARM inline assembly `LDM` or `STM` instructions, or VFP `FLDM` or `FSTM` instructions, but does split Thumb `LDM` and `STM` inline assembly instructions where possible.

This option has no significant benefit for cached systems, or for processors with a write buffer.

This option also has no benefit for systems with non-zero-wait-state memory, or for systems with slow peripheral devices. Interrupt latency in such systems is determined by the number of cycles required for the slowest memory or peripheral access. This is typically much greater than the latency introduced by multiple register transfers.

Narrow double constants to float constants

Select this option to change the type of unsuffixed floating-point constants from **`double`** (as specified by the ANSI/ISO C and C++ standards) to *unspecified*. In this context, *unspecified* means that uncast **`double`** constants and **`double`** constant expressions are treated as **`float`** when used in expressions with values other than **`double`**. This can sometimes improve the execution speed of a program.

Compile-time evaluation of constant expressions that contain such constants is unchanged. The compiler uses double-precision calculations, but the unspecified type is preserved. For example:

```
(1.0 + 1.0) // evaluates to the floating-point
            // constant 2.0 of double precision and
            // unspecified type.
```

In a binary expression that must be evaluated at run-time (including expressions that use the ?: operator), a constant of unspecified type is converted to `float`, instead of `double`. The compiler issues the following warning:

```
C2621W: double constant automatically converted to float
```

You can avoid this warning by explicitly suffixing floating-point constants that you want to be treated as `float`. You can turn this warning off with the `-Wk` compiler option.

——— **Note** ———

This behavior is not in accordance with the ANSI C standard.

————————

If the other operand in the expression has type `double`, a constant of unspecified type is converted to `double`. A cast of a constant of unspecified type to type T produces a constant of type T.

For example:

```
float f1(float x) { return x + 1.0; }        // Uses float add and is treated the same
                                             // as f2() below, a warning is issued.
float f2(float x) { return x + 1.0f;}        // Uses float add with no warning, with or
                                             // without -auto_float_constants.
float f3(double x) { return x + 1.0; }       // Uses double add, no special treatment.
float f4(float x) { return x + (double)1.0;} // Uses double add, no special treatment.
```

One ELF section per function

Select this option to generate one ELF section for each function in a source file. This option enables the linker to remove unused functions. This option increases code size slightly for some functions, but when creating code for a library, it can prevent unused functions being included at the link stage. This can result in the reduction of the final image size.

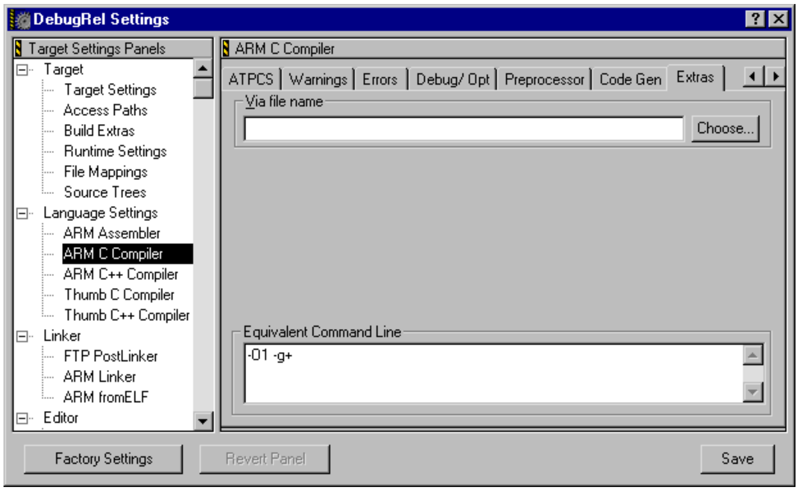
4. Click **Save** to save your changes.

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 *ADS Compiler, 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 9-4).
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 9-27).



-01 -g+

Figure 9-27 ARM 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 **Save** to save your changes.

9.6 Configuring linker settings

This section describes how to configure the ARM 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 postlinker you can use dependent subprojects or subtargets to select additional linkers or postlinkers. See *Working with multiple build targets and subprojects* on page 2-53 for more information.

You can use the following linker and postlinker options with the ARM tool chain:

FTP PostLinker

This panel is not used by CodeWarrior for the ARM Developer Suite.

Linker Use this panel to configure the ARM linker. See *Configuring the ARM linker* on page 9-62 for details.

fromELF Use this panel to configure the ARM 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 fromELF* on page 9-72 for details.

The ARM librarian

There is no configuration panel for armar. You can use armar to combine ELF object files into a library. See also the utilities chapter of the *ADS Compiler, Linker, and Utilities Guide* for detailed information on the ARM implementation of ar.

The batch runner

There is no configuration panel for the batch runner. See *Running batch files with the batch runner* on page 2-84 for more information.

For information on configuring the other tools in the ARM tool chain see:

- *Configuring assembler and compiler language settings* on page 9-31
- *Configuring the debugger* on page 9-77.

——— **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.

———————————

9.6.1 Configuring an FTP PostLinker

This panel is not used by CodeWarrior for the ARM Developer Suite.

9.6.2 Configuring the ARM linker

This section describes how to configure options for the ARM linker (armlink). It provides a general description of the options that you can configure through the CodeWarrior IDE. See the *ADS Compiler, Linker, and Utilities Guide* for a description of:

- the ARM 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* on page 9-62
- *Configuring linker options* on page 9-66
- *Configuring image layout* on page 9-68
- *Configuring linker listings* on page 9-69
- *Configuring linker extras* on page 9-71.

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 link or armar operation. See also*Working with multiple build targets and subprojects* on page 2-53 for information on configuring dependent subtargets and subprojects.

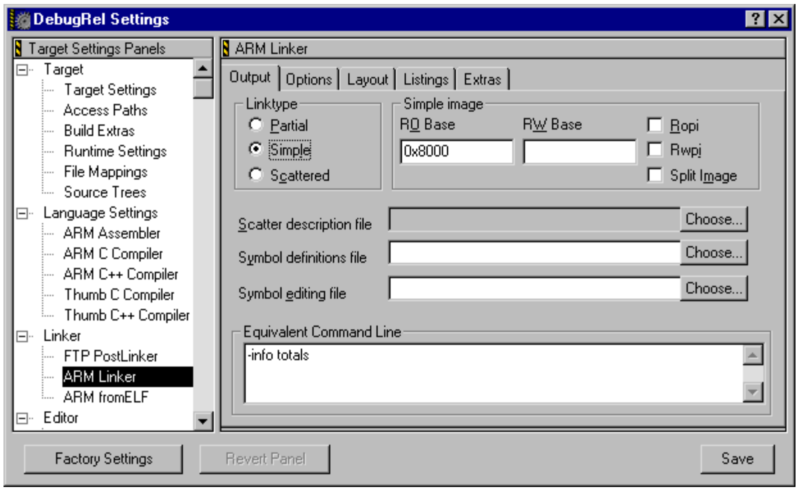
- A simple image that does not require a scatter-load description file to describe the structure of the image. This option provides an easy way to produce an executable ELF image. It gives limited control over the structure of the image.

- 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 linker chapter of the *ADS Compiler, 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 *ADS 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 9-4).
2. Click **ARM Linker** in the Target Settings Panels list and click the **Output** tab to display the configuration panel (Figure 9-28).



Linktype: ○ Partial [✓ Simple] ○ Scattered

RO Base: 0x8000

Equivalent Command Line: -info totals

Figure 9-28 ARM linker Output panel

3. Select the type of image you want to produce in the Link type group of options. Select one of:

Partial	Select this option to produce a partially linked ELF object file. No other options are available if you select this option.
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 9-68 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 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 `.scf`. You can add scatter-load description files to your project. See the linker chapter of the *ADS Compiler, 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 This text field sets both the load address and execution address of the region containing the RO section. If you do not enter a value, the value defaults to `0x8000`.

RW Base

This text field sets the execution addresses of the region containing the RW and ZI output sections. If you enter a value in this field, the linker creates an image with an execution view that contains two, possibly non-contiguous, regions:

- a region containing the RO output section
- a region containing the RW and ZI output sections.

If you enter a value for RW Base and select the **Split image** option, the linker creates an image that has two load regions in addition to two execution regions. In this case the value you enter for RW Base sets both the load address and the execution address of the region that contains the RW and ZI output sections.

Ropi Select this option to instruct the linker to make the execution region containing the RO output section position-independent. Usually, each read-only input section must be read-only position-independent. If this option is selected, the linker:

- checks that relocations between sections are valid
- ensures that any code generated by the linker itself, such as interworking veneers, is read-only position-independent.

Note

The ARM tools cannot determine if the final output image will be Read-Only Position-Independent until the linker finishes processing input sections. This means that the linker might emit ROPI error messages, even though you have selected the ROPI options for the compiler and assembler.

Rwpi Select this option to instruct the linker to make the execution region containing the RW and ZI output sections position-independent. If this option is not selected, the region is marked as absolute. Each writable input section must be read-write position-independent. If this option is selected, the linker:

- checks that the PI attribute is set on input sections to any read-write execution regions
- checks that relocations between sections are valid
- generates sb-relative entries in Region$$Table and ZISection$$Table.

This option requires a value for RW Base. A value of zero (0) is assumed if you do not specify one.

Note

The compiler does not force your writable data to be position-independent. This means that the linker might emit RWPI error messages, even though you have selected the RWPI options for the compiler and assembler.

Split Image

Select this option to split the default load region, which contains the RO and RW output sections, into two load regions:

- one containing the RO output section
- one containing the RW output section.

This option requires a value for RW Base. A value of zero (0) is assumed if you do not specify one.

5. Select a symbol definitions file, if required. See the description of the `-symbols` option in the *ADS Compiler, Linker, and Utilities Guide* for more information on symbol files.

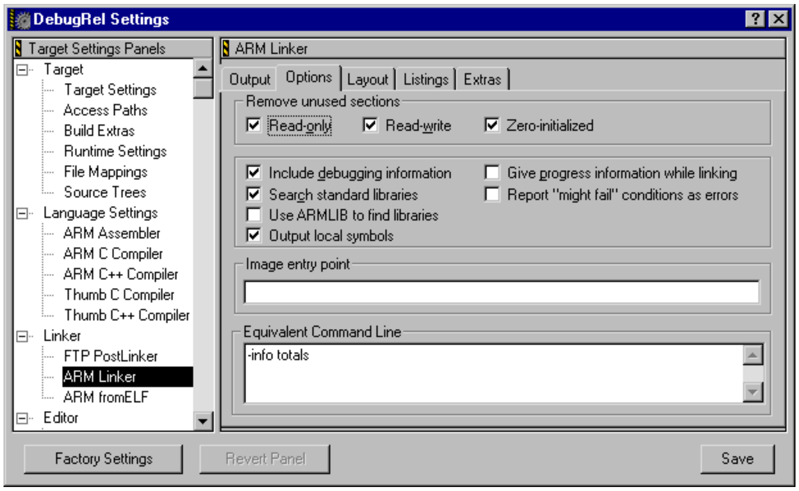
6. Select a symbol editing file, if required. See the description of the `-edit` option in the *ADS Compiler, Linker, and Utilities Guide* for more information on editing symbols in output objects.

7. Click **Save** to save 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 9-4).
2. Click **ARM Linker** in the Target Settings Panels list and click the **Options** tab to display the configuration panel (Figure 9-29).



Remove unused sections
☑ Read-only ☑ Read-write ☑ Zero-initialized

☑ Include debugging information
☑ Search standard libraries
☐ Use ARMLIB to find libraries
☑ Output local symbols
☐ Give progress information while linking
☐ Report "might fail" conditions as errors

Image entry point

Equivalent Command Line
-info totals

Figure 9-29 ARM Linker Options panel

3. Select unused section elimination options, as required:

   Remove unused sections

   Select one or more of these options to remove unused sections from the image:

   - **Read-only**
   - **Read-write**
   - **Zero-initialized.**

   See the description of unused section elimination in the linker chapter of the *ADS Compiler, Linker, and Utilities Guide* for detailed information on when a section is considered to be unused.

Caution

You must take care not to remove interrupt handlers when you remove unused read-only sections. There are a number of ways to do this, including:

- using `-keep` directly in the command line text field
- ensuring that the section containing the interrupt handlers is placed first
- using the `ENTRY` assembler directive.

See the linker chapter in the *ADS Compiler, Linker, and Utilities Guide* for more information on unused section elimination.

4. Select additional linker options, as required. The following options are available:

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.

If this option is not selected, the linker discards any debug input section 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.

Use ARMLIB to find libraries

Select this option to search the paths defined in the ARMLIB environment variable for the ARM standard C libraries, instead of the system paths defined in the Access Paths panel. See *Configuring access paths* on page 9-11 for more information on access paths.

Output local symbols

Select this option to instruct the linker to add local symbols to the output symbol table when producing an executable image.

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.

5. 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 *ADS Compiler, Linker, and Utilities Guide* for details.

6. Click **Save** to save 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** ———

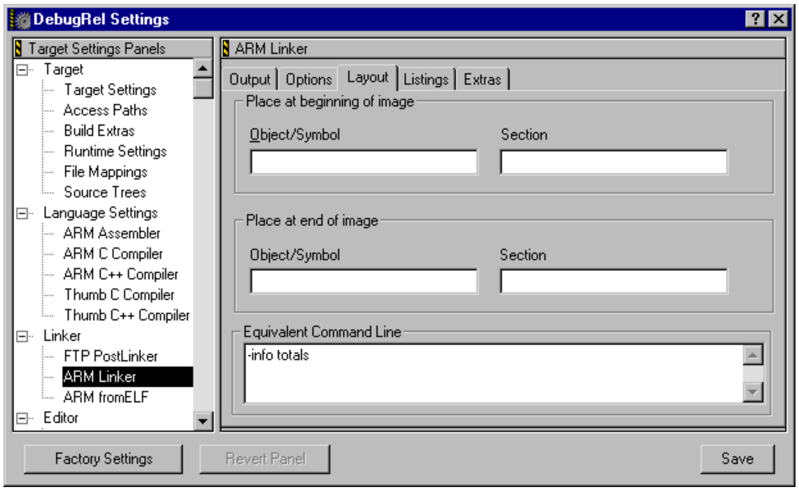
You cannot use these options to override the basic attribute sorting order for sections in output regions. The basic sorting rules place RO sections before RW sections, and place both RO and RW sections before ZI sections. This means, for example, that you cannot use the image layout options to place an RW section before an RO section in an output region.

You can use scatter-loading to specify more complex ordering of sections. See the Linker chapter in the *ADS Compiler, 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 9-4).
2. Click **ARM Linker** in the Target Settings Panels list and click the **Image Layout** tab to display the configuration panel (Figure 9-30).



```
-info totals
```

Figure 9-30 ARM 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 linker chapter of the *ADS Compiler, Linker, and Utilities Guide* detailed information on specifying first and last sections.

4. Click **Save** to save 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 9-4).

2. Click **ARM Linker** in the Target Settings Panels list and click the **Listings** tab to display the configuration panel (Figure 9-31).

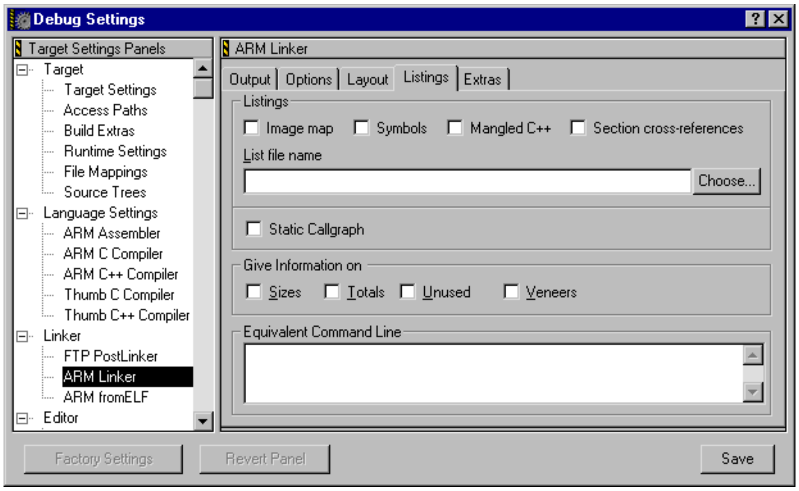


Figure 9-31 ARM Linker Listings panel

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. When you make your project, the listing text file is displayed in an editor window.

   If you do not enter a filename the listing information is displayed in a Message window.

5. Select the **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 *ADS Compiler, 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 9-66 for more information on unused section elimination.
Veneers	Gives details of the linker-generated veneers. See the linker chapter in the *ADS Compiler, Linker, and Utilities Guide* for more information on linker-generated veneers, such as interworking veneers.

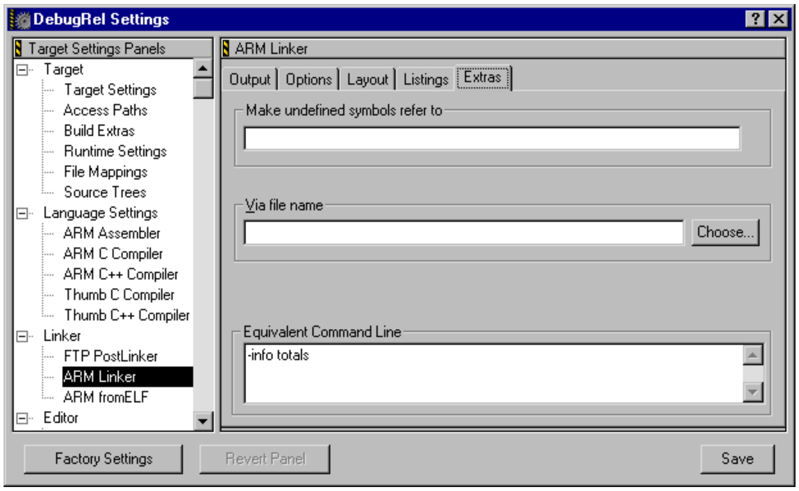
7. Click **Save** to save your changes.

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 9-4).

2. Click **ARM Linker** in the Target Settings Panels list and click the **Extras** tab to display the configuration panel (Figure 9-31 on page 9-70).



```
-info totals
```

Figure 9-32 ARM Linker Extras panel

3. Enter a symbol to match to each reference to an undefined symbol in your code.

   The symbol must be both defined and global, otherwise it will appear in the list of undefined symbols, and 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. 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 linker chapter in the *ADS Compiler, Linker, and Utilities Guide* for more information on using via files with the linker.

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

9.6.3 Configuring fromELF

The 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 9-8 for more information.

In addition to calling fromELF as a post-linker, you can also call it as a third-party debugger. This enables you to use fromELF a second time, for example, to generate a disassembled code listing from your converted binary output. See *Choosing a debugger* on page 9-78 for more information.

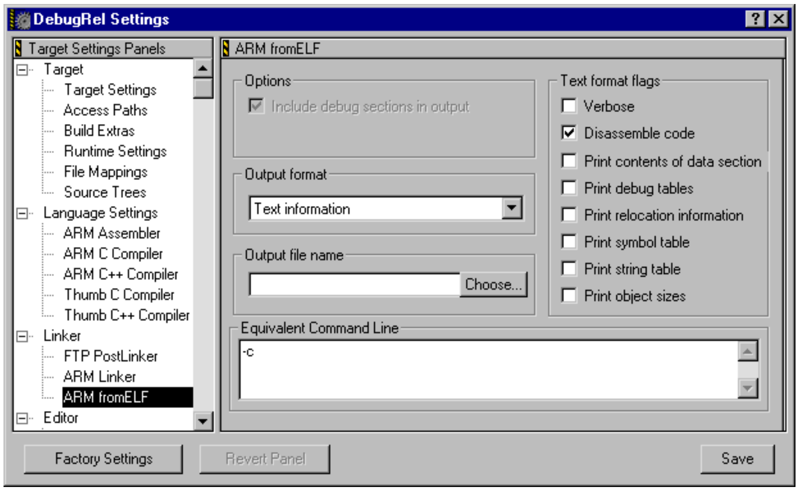
Note

The options you configure in the ARM fromELF panel apply only when you call fromELF as a post-linker, not when you call it as a third-party debugger.

To configure the fromELF utility:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Click **ARM fromELF** in the Target Settings Panels list to display the fromELF configuration panel (Figure 9-33).



Options
- [✓] Include debug sections in output

Output format: Text information

Output file name:

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: -c

Figure 9-33 ARM fromELF panel

3. Select **Include debug sections in output** to ensure that debug table information is included in the fromELF output.

4. Click the **Output format** pop-up menu and select the output format you want. Either:

   - Select from the binary output options. See the Toolkit Utilities chapter of the *ADS Compiler, Linker, and Utilities Guide* for details.

- Select Text information to extract a text file of information on the output image. Select one or more text format flags to choose the information you want. These flags also control how source files are disassembled when you use the **Disassemble** command. See *Disassembling code* on page 2-81 for more information.

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 **Save** to save your changes. When you make your project, the CodeWarrior IDE calls fromELF to process the output.

9.7 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 8-14 for information on setting global editor preferences.

9.7.1 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 *Syntax Coloring* on page 8-22 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 9-4).
2. Click **Custom Keyword** in the Target Settings Panels list to display the configuration panel (Figure 9-34)

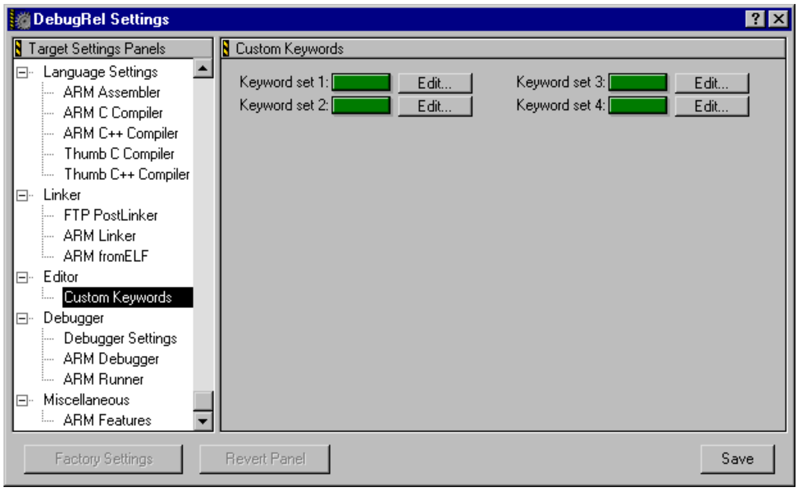


Figure 9-34 Custom Keywords settings panel

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 **Save** to save your changes.

9.8 Configuring the debugger

You can configure the CodeWarrior IDE to call any of the ARM debuggers to load and debug your images. This section describes how to configure the ARM debuggers to debug and run executable images built from CodeWarrior IDE projects. See Chapter 3 *Working with the ARM Debuggers* for more information on how the CodeWarrior IDE interacts with the ARM debuggers. See the *ADS Debuggers Guide* for detailed information on working with the ARM debuggers.

For information on configuring the other tools in the ARM tool chain see:

- *Configuring assembler and compiler language settings* on page 9-31
- *Configuring linker settings* on page 9-61.

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:

- *Debugger settings* on page 9-77
- *Configuring the ARM Debugger* on page 9-77
- *Configuring the ARM Runner* on page 9-87.

9.8.1 Debugger settings

The Debugger settings panel is not used by the ARM version of the CodeWarrior IDE.

9.8.2 Configuring the ARM Debugger

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. Depending on the debugger, you can provide startup and configuration options, or specify a script file to call when the debugger is executed.

In addition, you can use this panel to specify a third-party debugger, or other post-processing tool, in place of the ARM debuggers. The following sections describe:

- *Choosing a debugger* on page 9-78
- *Configuring ADW* on page 9-80
- *Configuring armsd* on page 9-83
- *Specifying arguments for your executable image* on page 9-86.

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. You can select any ARM debugger, or you can specify a third-party debugger with the **Other** option.

——— **Note** ———

Use the **Other** option in this panel in preference to the **Use Third Party debugger** option Build Extras panel.

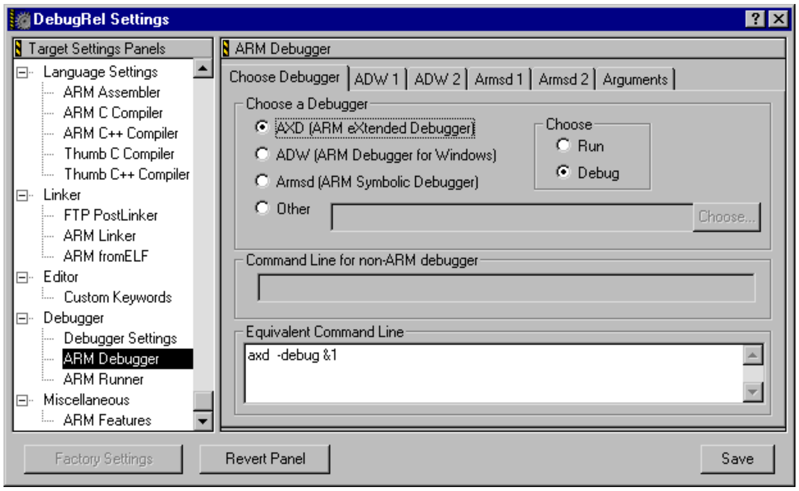
————————————

In addition, you can use the **Other** option to select any post-processing tool for your output images. You can use this, for example, to run the ARM fromELF utility twice during a build so that you can generate a disassembled listing of a plain binary output file:

1. Configure fromELF as a post-linker to convert your output image to ROMable binary format (see *Configuring target settings* on page 9-8 and *Configuring fromELF* on page 9-72)
2. Call fromELF as a third-party debugger to generate a disassembled code listing of your output binary.

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 9-4).
2. Click **ARM Debugger** in the Target Settings Panels list and click the **Choose Debugger** tab to display the configuration panel (Figure 9-35 on page 9-79).



Choose a Debugger: [✓ AXD (ARM eXtended Debugger)] ADW (ARM Debugger for Windows) Armsd (ARM Symbolic Debugger) Other

Choose: Run [✓ Debug]

Command Line for non-ARM debugger

Equivalent Command Line

```
axd -debug &1
```

Figure 9-35 Choose debugger panel

3. Choose the debugger you want to use when you select **Debug** from the **Project** menu. To select an ARM debugger, click one of the **AXD**, **ADW**, or **Armsd** radio buttons and configure the debugger in the appropriate panel. For more information on configuring ADW or armsd see:

   - *Configuring ADW* on page 9-80
   - *Configuring armsd* on page 9-83.

   ——— **Note** ———

   AXD does not have a configuration panel

   ————————

   See also the *ADS Debuggers Guide* for detailed information on using the ARM Debuggers.

   To select a third-party debugger:

   a. Click **Other**.

   b. Click **Choose…** and select the third-party debugger from the standard file dialog.

   c. Enter the debugger command-line arguments in the Command-line for non-ARM debugger field.

——— **Note** ———

Third-party debuggers must be capable of loading and debugging an ARM ELF/DWARF2 image.

————————

Configuring ADW

If you have configured the CodeWarrior IDE to use ADW as your debugger you can configure a number of ADW options from the CodeWarrior IDE.

Note

Use the ADW configuration dialog to configure other ADW options that are not available from the CodeWarrior IDE, such as the byte order used for target memory, and ARMulator™ clock speed.

Use the following panels to configure ADW options:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).

2. Click **ARM Debugger** in the Target Settings Panels list and click the **ADW1** tab to display the first configuration panel (Figure 9-36).

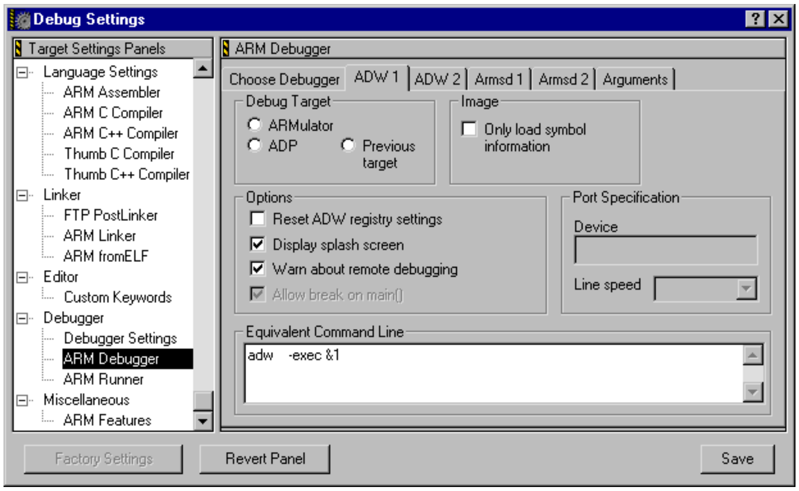


Figure 9-36 ADW1 panel

3. Select the debug target for the debugger:

ARMulator

Select **ARMulator** to target an ARMulator processor.

ADP Select **ADP** 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.

Previous target

Select **Previous target** if you want the debugger to start with the debug target configuration that was used for the most recent debugging session.

——— **Note** ———

Targeting Multi-ICE®

You should use this option if you are targeting Multi-ICE:

1. Start ADW and select **Configure Debugger** from the **Options** menu to configure the debugger to target Multi-ICE.
2. Select the **Previous target** option from the ADW1 configuration panel to configure the CodeWarrior IDE to restart the debugger with Multi-ICE.

---

3. Enter an expression to select the target communication method if you have selected an ADP debug target in the previous step. 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 prefix `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.

4. Select **Only load symbol information** if you want to load only debugging information from an image file. You can use this option when you do not need to download an image because it already exists on your target, for example, in Flash.

5. Select other debug options:

Reset ADW registry settings

Select this option to reset the ADW Windows registry entries on startup.

Display splash screen

Select this option to display the ADW splash screen on startup.

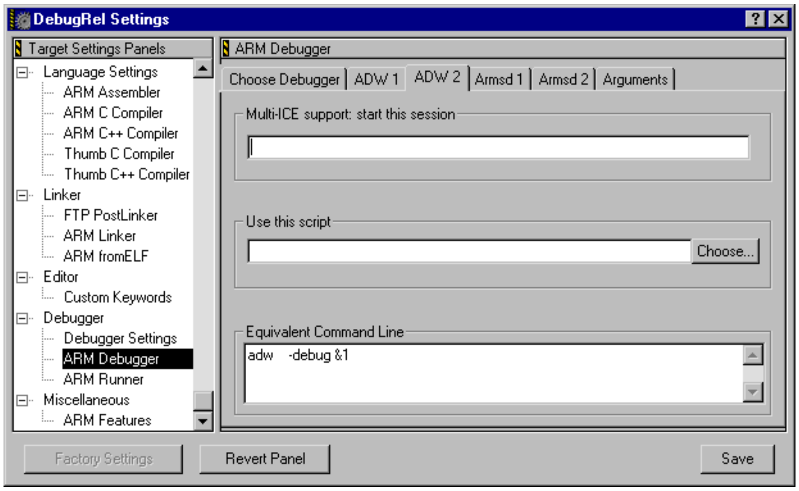
Warn about remote debugging

Select this option to enable the warning given when ADW is started with an ADP remote debug target.

Allow break on main()

Select this option to allow ADW to insert a breakpoint on `main()`. This option is selected by default. Deselect it if you want to run your executable images until they reach an explicit breakpoint, or until program termination if there are no breakpoints.

6. Click **ARM Debugger** in the Target Settings Panels list and click the **ADW 2** tab to display the second configuration panel (Figure 9-37).



```
adw  -debug &1
```

Figure 9-37 ADW2 panel

7. Enter Multi-ICE session and script file information if required:

Multi-ICE Support: start this session

Use this field to specify an ADW *session* name. You can use this option to save ADW configuration settings in the windows registry:

- If you specify a new session name, ADW creates a new named session and saves the configuration information for the current debug session in the Windows registry when you exit ADW.
- If you specify a previously used session name, ADW is configured using the information in the named session.

This option is useful for saving and restoring multiple configurations for use with Multi-ICE, or in any other case where you want to restore your previous ADW configuration.

Use this script

Use this field to specify a *script* file containing additional startup commands for ADW. A script file is a text file that contains a sequence of commands in the ADW/armsd command language. For example, the script file can contain commands to set breakpoints, load an image to memory, and execute to a specified point in the image.

Enter the full pathname to a script file, or click **Choose…** and select a script file from the standard file dialog box.

8. Use the Arguments tab to enter command-line arguments to be passed to the executable image you are debugging. See *Specifying arguments for your executable image* on page 9-86 for more information.

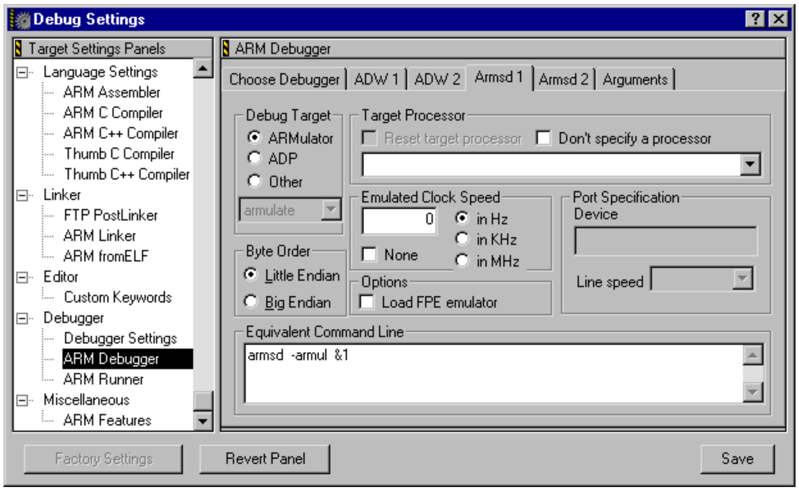
9. Click **Save** to save your settings.

Configuring armsd

If you have configured the CodeWarrior IDE to use armsd as your debugger (see *Choosing a debugger* on page 9-78), use the 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 9-4).

2. Click **ARM Debugger** in the Target Settings Panels list and click the **Armsd1** tab to display the first configuration panel (Figure 9-38).



Debug Settings

Target Settings Panels
- Language Settings
  - ARM Assembler
  - ARM C Compiler
  - ARM C++ Compiler
  - Thumb C Compiler
  - Thumb C++ Compiler
- Linker
  - FTP PostLinker
  - ARM Linker
  - ARM fromELF
- Editor
  - Custom Keywords
- Debugger
  - Debugger Settings
  - ARM Debugger
  - ARM Runner
- Miscellaneous
  - ARM Features

ARM Debugger

Choose Debugger | ADW 1 | ADW 2 | Armsd 1 | Armsd 2 | Arguments

Debug Target: [✓ ARMulator] ADP Other; armulate

Target Processor: Reset target processor ☐ Don't specify a processor ☐

Emulated Clock Speed: 0 [✓ in Hz] in KHz in MHz; None ☐

Port Specification: Device; Line speed

Byte Order: [✓ Little Endian] Big Endian

Options: Load FPE emulator ☐

Equivalent Command Line:

```
armsd -armul &1
```

Factory Settings | Revert Panel | Save

Figure 9-38 Armsd1 panel

3. Select the debug target for the debugger:

   ARMulator
   : Select **Armulator** to target an ARMulator processor.

   ADP
   : Select **ADP** 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.

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.

   Emulated Clock Speed
   : Select a clock speed for the ARMulator. This option is valid only if you you are using an `armsd.map` file. The `armsd.map` file must be located in the same directory as armsd. See *ADS Debug Target 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.

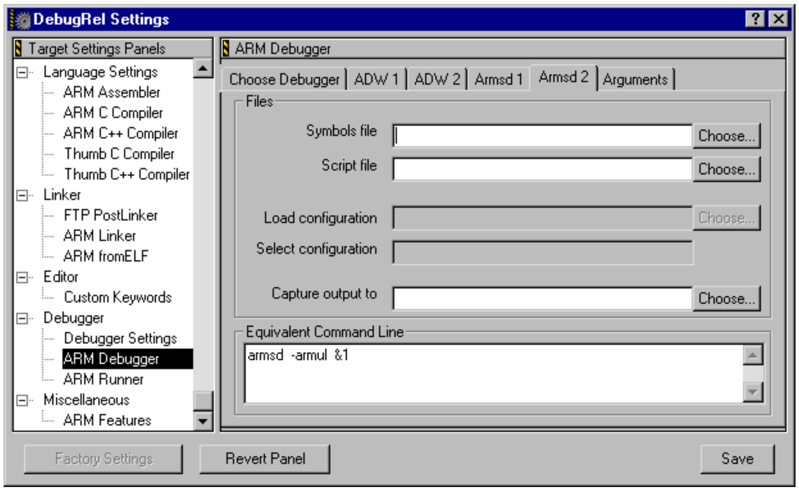
     `e=`*`id`* selects ethernet communication. *`id`* is the ethernet address of the target board.

For serial and serial/parallel communications, you can prefix `,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 9-39).



```
armsd -armul &1
```

Figure 9-39 Armsd2 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.

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.

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 *ADS Debuggers Guide* for more information.

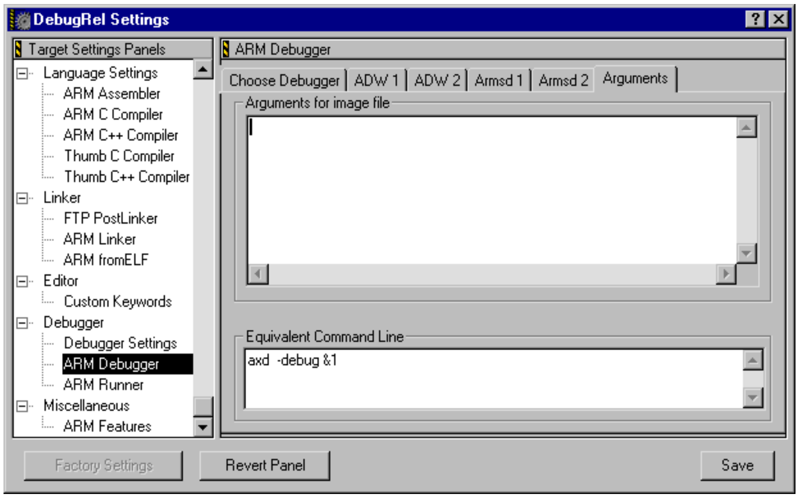
Capture output to

Enter a filename to which output information from the debuggee is written. Alternatively, click **Choose…** to select an output file from the standard file dialog box.

7. Click **Save** to save 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 9-4).
2. Click **ARM Debugger** in the Target Settings Panels list.
3. Click the **Arguments** tab to display the arguments configuration panel (Figure 9-40 on page 9-86). Use this panel to enter any command-line arguments required by your executable image.



axd -debug &1

Figure 9-40 Arguments panel

4. Click **Save** to save your settings.

9.8.3 Configuring the ARM Runner

The term *ARM Runner* refers to the ARM debugger that is called to execute, rather than debug, an image file.

The ARM 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, or a third-party debugger, 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 ARM Debugger panel. See *Configuring the ARM Debugger* on page 9-77 for information.

9.9 Configuring Miscellaneous settings

This section describes the following miscellaneous configuration options:

- *ARM Features* on page 9-88.

9.9.1 ARM Features

This panel is required only if you are using versions of the ARM Developer Suite that are feature-restricted through FLEX*lm* license management software. The ARM Features panel enables you to select the feature set supplied with your version of ADS. This panel enables you to select the feature set in use if you are moving a project from one restricted toolkit to another restricted toolkit with a different feature set.

To configure the feature set:

1. Display the Target Settings window for the build target you want to configure (see *Displaying Target Settings panels* on page 9-4).
2. Click **ARM Features** in the Target Settings Panels list to display the configuration panel (Figure 9-41).

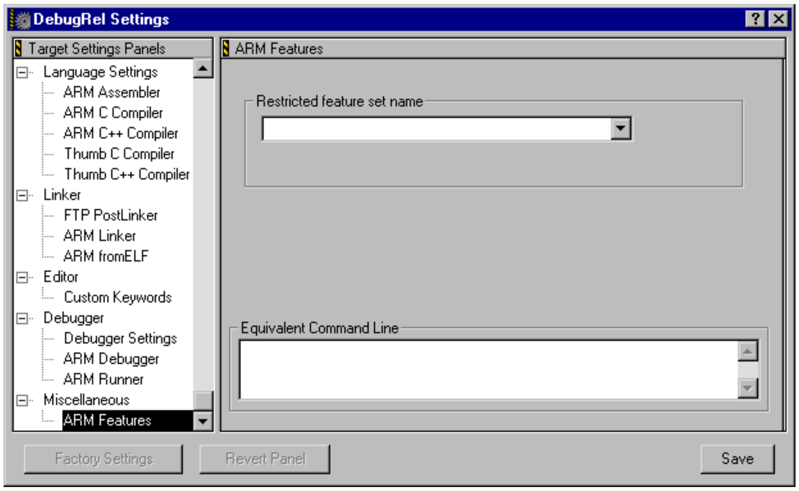


Figure 9-41 ARM Features panel

3. Select the appropriate feature set.
4. Click **Save** to save your changes.

Chapter 10
Using the CodeWarrior IDE with Version Control Systems

This chapter explains how to use the CodeWarrior IDE version control integration facilities to control your source code. It contains the following sections:

10.1 About version control systems

A revision control or *version control system* (VCS) enables you to maintain a database of your source code, and then check files in or out of the database. Version control systems can help you track code changes, particularly when more than one person is working on a software project.

Note

See *Configuring CodeWarrior for complex or multi-user projects* on page 2-51 for important information on using CodeWarrior in multi-user environments.

10.1.1 Commercially available VCS plug-ins

The CodeWarrior IDE plug-in architecture supports a variety of version control systems.

Plug-ins are available for the following version control systems:

- CVS
- Visual SourceSafe
- Clearcase
- Perforce.

To see the latest list of available plug-in tools, visit the Metrowerks web site at:
`http://www.metrowerks.com`

10.2 Activating VCS

This section describes how to install and activate a version control system so that it can be used with the CodeWarrior IDE.

10.2.1 VCS plug-in software

The installation instructions for your VCS plug-in software will depend on which VCS software you are using. This section gives general instructions that apply to most VCS systems.

Installing VCS plug-in software

To install most version control system plug-in software:

1. Exit the CodeWarrior IDE if it is running.
2. Copy your VCS plug-in to the `{Compiler}\Plugins` directory.

   Note

   You must also follow any installation instructions that accompany the VCS software you plan to use. If you encounter any problems, contact the VCS software vendor for assistance.

3. Restart the CodeWarrior IDE.

This is usually sufficient to make the VCS software available for use.

10.2.2 Activating VCS software

You must configure VCS options separately for each project that you want to use your version control system. This section gives general instructions for activating version control for a project.

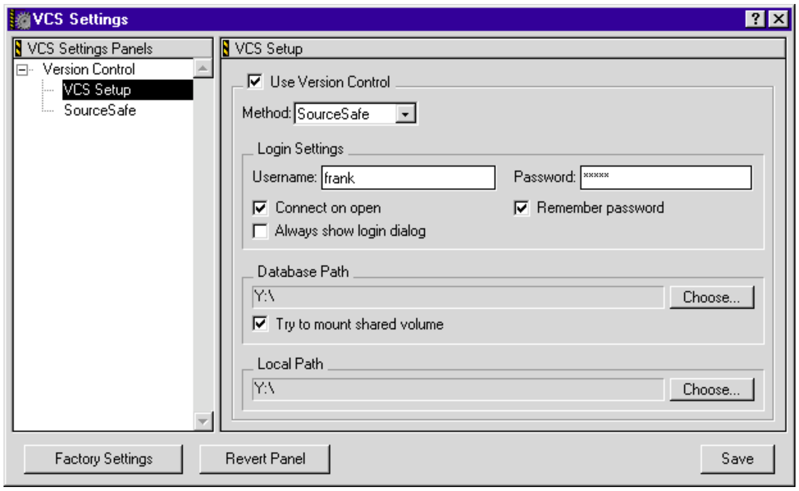
Note

You must also follow any activation instructions that accompany the VCS software you plan to use. If you encounter any problems, contact the VCS software vendor for assistance.

Configuring your VCS settings

To configure your VCS settings and activate VCS:

1. Ensure that the project you want to configure is the currently active window.
2. Select **Version Control Settings…** from the **Edit** menu. The VCS Settings window is displayed (Figure 10-1).



VCS Settings

VCS Settings Panels
- Version Control
  - VCS Setup
  - SourceSafe

VCS Setup

☑ Use Version Control

Method: SourceSafe

Login Settings

Username: frank   Password: *****

☑ Connect on open   ☑ Remember password

☐ Always show login dialog

Database Path

Y:\

☑ Try to mount shared volume

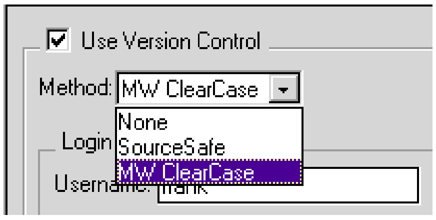
Local Path

Y:\

Figure 10-1 The VCS settings panel

3. Select the **Use Version Control** checkbox to activate version control.
4. Select a version control system from the **Method** menu.

   The CodeWarrior IDE supports several different types of VCS. If your VCS software is correctly installed, its name is displayed in the **Method** list, as shown in Figure 10-2.



☑ Use Version Control

Method: MW ClearCase

None
SourceSafe
MW ClearCase

Login

Username: frank

Figure 10-2 VCS Method pop-up list.

5. Enter your VCS user name in the Username field.
6. Enter your password (optional).

7. Set any additional Login Settings options you require. The following options are available:

   Connect on Open

   If this option is selected, CodeWarrior connects to the VCS database when you open this project.

   Always Show Login Dialog

   If this option is selected the login window is displayed every time you connect to the VCS database.

   Remember Password

   If this option is selected your password is saved after you connect to the database for the first time. You will not be required to enter your password each time you connect.

8. Click the **Choose** button next to the Database Path field to select the VCS database you want to access. The setting of the Database Path field depends on which VCS software you are using. See your VCS software documentation for more information.

9. Click the **Choose** button next to the Local Path field to select the destination folder where your local files will be stored. The setting of the Local Path field depends on which VCS software you are using. See your VCS software documentation for more information.

10. Click **Save** to save your settings. The CodeWarrior IDE uses the settings you have specified to connect to the VCS database.

——— **Note** ———

Some VCS software might have additional setup requirements. If you encounter any problems, contact the VCS software vendor for assistance.

10.3 Using your VCS from the CodeWarrior IDE

The CodeWarrior IDE provides several ways to access common VCS operations and view status and log information. These include:

- *Using the Version Control Login window* on page 10-6
- *Performing common VCS operations* on page 10-7.

10.3.1 Using the Version Control Login window

Many version control systems require you to log in before you can use the system. The CodeWarrior IDE provides the Version Control Login window to support systems that require password authentication.

The CodeWarrior IDE displays the Version Control Login window if you are not already logged in to your VCS and:

- You perform an operation that requires CodeWarrior to communicate with the VCS, such as updating the status of a file.
- You log in manually.
- You configure your project to connect each time the project is opened. See *Configuring your VCS settings* on page 10-4 for more information.

Logging in to your VCS

To log into your VCS:

1. Either:
   - perform an operation that will cause the CodeWarrior IDE to communicate with your VCS
   - select **Connect** from the **VCS** menu to connect manually, if this menu item is supported by your VCS software.

   The CodeWarrior IDE displays the Version Control Login window.
2. Enter your VCS username and password, if required by your system.
3. Click **OK** to log in, or **Cancel** to stop.

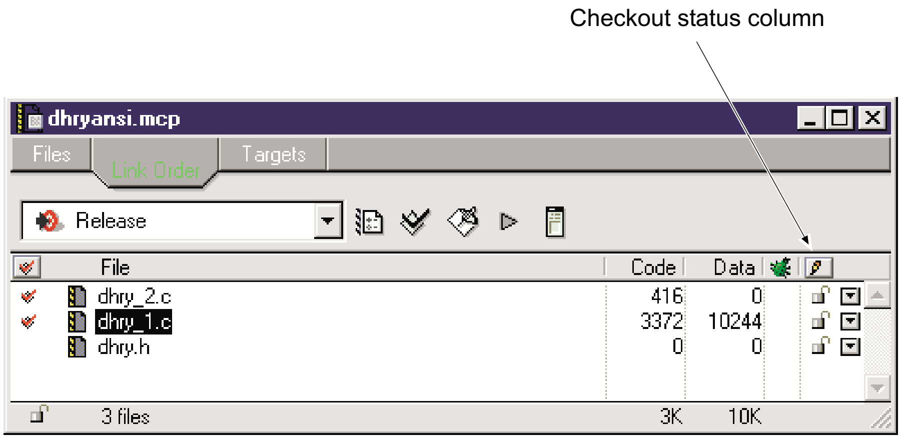
Logging out of your VCS

To log out of your VCS select **Disconnect** from the **VCS** menu.

10.3.2 Performing common VCS operations

When version control is active for a project, the CodeWarrior IDE makes the following additions to its graphical interface:

- A Checkout Status column is displayed in the project window. Figure 10-3 shows an example.
- A **VCS** menu is added to the menu bar. See Figure 10-4 on page 10-9 for an example.
- A **VCS** pop-up menu is added to editor windows for the project. See Figure 10-5 on page 10-11 for an example.



File	Code	Data
dhry_2.c	416	0
dhry_1.c	3372	10244
dhry.h	0	0

Figure 10-3 Checkout Status Column

You can use the VCS interface items to perform the most common VCS operations, such as determining file status, checking in files, and checking out files, from within both the project window, and the editor window.

——— **Note** ———

Other operations

Other operations might be available, depending on which revision control system plug-in you use. See the documentation that accompanies the VCS software you are using for more information.

The following sections describe how to perform the most common VCS operations from within the CodeWarrior IDE project and editor windows:

- *Viewing and synchronizing the VCS status of files*

- *Working from a project window* on page 10-9
- *Working from an editor window* on page 10-11
- *Viewing VCS messages* on page 10-12.

Viewing and synchronizing the VCS status of files

If you have configured your project to use a VCS, the CodeWarrior IDE uses icons to represent the current checkout status, or file permission, of the files in your project. When you change the checkout status of a file in the CodeWarrior IDE, the icon updates to reflect the change.

Each file in a project can have a different permission setting. The CodeWarrior IDE displays file status icon for a file:

- In the Checkout Status column of the project window. See Figure 10-3 on page 10-7 for an example.
- As the icon for the **VCS Pop-up** menu in the editor window for the file. See Figure 10-3 on page 10-7 for an example.

Table 10-1 shows the most common status indicator icons. The operations that you can perform on a file depend on the current status of the file, and the type of VCS system you use.

Table 10-1 VCS status icons

If the icon is…		Then…
✏	Checked out	You can edit the file and add your changes to the revision control database.
✗	Checked in	You cannot edit the file. It is part of the revision control database.
✏	Writable	You can edit the file, but you cannot add your changes to the revision control database because the file was not properly checked-out for modification.
🔓	Unlocked	The file can be edited. It is not checked into a revision control database.
🔒	Locked	You cannot edit the file. It is *not* part of the revision control database. You might not have the access privileges necessary to access the file, or someone might have locked the file.

Most version control systems provide a command to synchronize the status of a local copy of a file with the status of the file in the version control system. To perform a synchronize status command either:

- click the **Checkout Status** icon at the top of the column
- select **Synchronize Status** from the **VCS** menu.

The status of your local files is compared and synchronized with the status of the files in the version control database.

Working from a project window

You can perform many common VCS operations from the project window by using the **VCS** menu (Figure 10-4).

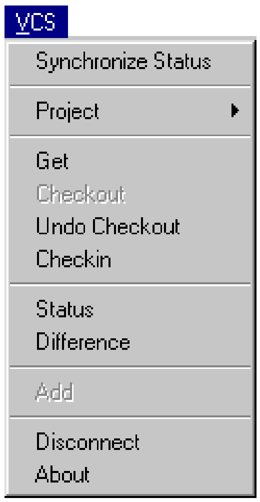


Figure 10-4 VCS menu

The **VCS** menu enables you to perform **Get**, **Checkout**, **Checkin**, and other common VCS operations. To perform a VCS operation on all the files in a project:

1. Ensure that the project window is the currently active window.
2. Select the command from the **VCS** menu.

——— **Note** ———

- The menu items available in the **VCS** menu might vary, depending on the VCS operations that are supported by your VCS software. See the documentation that accompanies your VCS software for more information.

- VCS plug-ins can assign different names to these operations. See the documentation that accompanies your VCS software to determine which operations are supported.

The following VCS menu commands are typical:

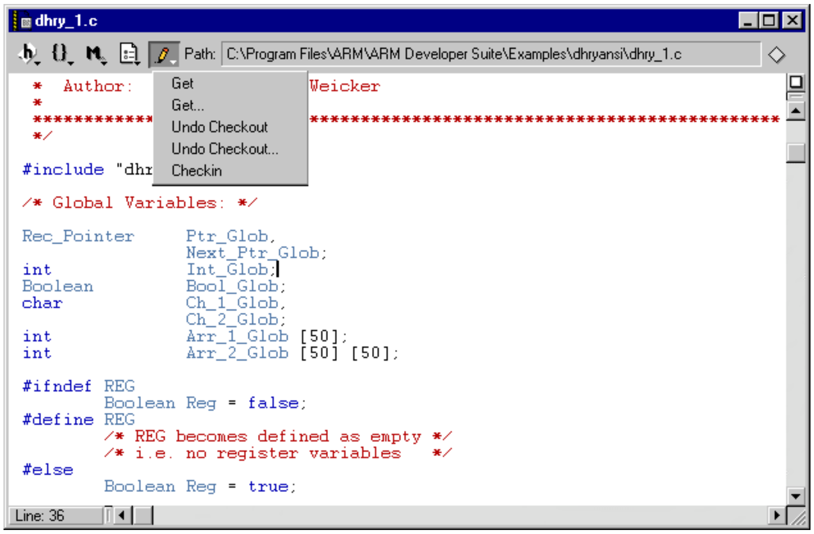
Synchronize Status	Updates the VCS Status column in the project window by examining each project file's status and updating its information. See *Viewing and synchronizing the VCS status of files* on page 10-8 for more information.
Project	Contains a submenu of VCS commands that enable you to perform **Get**, **Checkout**, **Undo Checkout**, **Checkin**, **Status**, and **Add** operations on project files themselves.
Recursive	Contains a submenu of VCS commands that enable you to perform recursive operations in some version control systems.
Get	Retrieves a copy of the file without checking it out of the project database.
Checkout	Checks out files for modification. Depending on your version control software, checkout can be exclusive or non-exclusive.
Undo Checkout	Cancels a checkout and discard all changes.
Checkin	Returns a modified file to the database and relinquishes the checkout.
History	Displays a modification history of a project or file.
Status	Displays the status of a file.
Properties	Displays database information about a project or file.
Comment	Changes a comment for a specific version of a project or file.
Label	Assigns a label to a project or file.
Add	Adds a file to the database.
Connect or Disconnect	Connects or disconnects you from the project database, depending on the current open status.
About	Displays VCS plug-in copyright and version information.

Variables

Displays the VCS user variable settings in an editor window.

Working from an editor window

The **VCS** pop-up menu (see Figure 10-5 on page 10-11) is displayed in editor windows when a VCS is activated. The icon that represents the **VCS** pop-up menu indicates the current permission setting for the file. For more information on permission settings, see *Viewing and synchronizing the VCS status of files* on page 10-8.


dhry_1.c
Path: C:\Program Files\ARM\ARM Developer Suite\Examples\dhryansi\dhry_1.c
Get
Get...
Undo Checkout
Undo Checkout...
Checkin
Line: 36

```
 *  Author:            Weicker
 *
 ************            ************************************************
 */

#include "dhr

/* 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];

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

Figure 10-5 VCS Pop-Up Menu

The VCS pop-up menu enables you to perform a subset of the available VCS commands on the file you are currently editing. The VCS operations you can perform depend on the current status of the file, and the operations supported by your VCS.

To perform a VCS operation on a source file:

1. Open the file in the CodeWarrior editor.
2. Click the **VCS** pop-up menu.

Note

The button that represents this menu changes to reflect the current status of the file. See Table 10-1 on page 10-8 for more information.

A VCS menu is displayed that contains menu items for VCS operations that are supported by your VCS system. See Figure 10-5 for an example.

See the documentation that accompanies your revision control system for more information on supported VCS operations.

3. Select an item from the **VCS** menu to execute a VCS operation. If the menu item contains an ellipsis character (…), a dialog box is displayed that enables you to customize the operation before you execute it.

   The following VCS pop-up operations are typical:

Unlock	Changes the lock on the file (if possible), enabling it to be writable.
Add	Adds the file to the revision control database.
Get	Retrieves a fresh copy of the file from the revision control database.
Checkout	Checks the file out from the revision control database for modifications.
Undo Checkout	Discards any changes made to the file, and instructs the revision control database to cancel the checkout of the file.
Checkin	Instructs the revision control database to accept the file with the changes that have been made to it.
Make Writable	Makes the file writable. Depending on your VCS, you might not be able to be check the modified version into the version control database.

Viewing VCS messages

The CodeWarrior IDE uses a message window to display a log of revision control messages. See *Using the message window* on page 3-15 for details of how to use the controls in the window.

Appendix A
Perl Scripts

This appendix describes how to use the CodeWarrior IDE Perl support. It describes how to install and configure the Perl plug-ins, and how to configure your project to recognize and run Perl scripts. It contains the following sections:

- *Perl software plug-ins* on page A-2
- *Configuring a prefix file* on page A-4
- *Using Perl scripting* on page A-6.

A.1 Perl software plug-ins

The CodeWarrior IDE uses software plug-ins to process Perl scripts. This section describes how to install the Perl plug-ins for Windows.

A.1.1 Installing Perl software plug-ins

The CodeWarrior IDE Perl plug-ins are available in the `MWPerlWin.zip` zip archive on your installation CD. To install the plug-ins:

1. Open the `MWPerlWin.zip` archive in a zip extraction utility such as WinZip.
2. Ensure that your zip extraction utility is configured to recreate the directory structure of the archive.
3. Extract the contents of the zip archive to the `bin\plugins` subdirectory of your ADS installation directory. If you have installed ADS in its default location, this will be `c:\Program Files\ARM\ADSv1_1\Bin\plugins`.
4. Start the CodeWarrior IDE. The Perl plug-ins are available.

A.2 Configuring your project for Perl

This section describes how to configure the CodeWarrior IDE to recognize and process Perl scripts, and how to configure a prefix file that is executed before each Perl script in your project.

A.2.1 Configuring file mappings

If you want to add Perl files with a standard `.pl` filename extension to your project you must configure the File Mappings panel to associate `.pl` files with the MW Perl plug-in compiler. To configure file mappings:

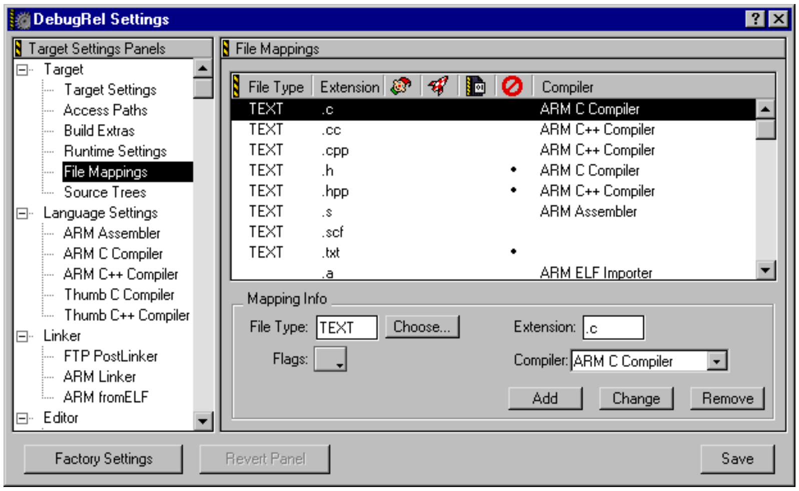
1. Open the project window for your project.

2. Select the build target you want to configure.

   Note

   You must configure File Mappings separately for each build target in your project.

3. Select ***target_name*** **Settings…** from the **Edit** menu and select File Mappings from the list of Target Settings Panels. The CodeWarrior IDE displays the File Mappings panel (Figure A-1).



File Type	Extension					Compiler
TEXT	.c					ARM C Compiler
TEXT	.cc					ARM C++ Compiler
TEXT	.cpp					ARM C++ Compiler
TEXT	.h				•	ARM C Compiler
TEXT	.hpp				•	ARM C++ Compiler
TEXT	.s					ARM Assembler
TEXT	.scf					
TEXT	.txt				•	
	.a					ARM ELF Importer

Mapping Info

File Type: TEXT

Flags:

Extension: .c

Compiler: ARM C Compiler

Figure A-1 File mappings configuration panel

4. Select an existing file mapping, such as the ARM C compiler mapping.

5. Change the Extension field to `.pl` and select MW Perl from the Compiler pop-up menu.

Note

If the MW Perl menu item is not displayed in the pop-up menu, check that you have correctly installed the Perl plug-in software. See *Perl software plug-ins* on page A-2 for more information.

6. Click **Add** to add the filename extension to the File Mappings list.
7. Click **Save** to save your changes.

A.2.2 Configuring a prefix file

You can use the Perl target settings panel to specify a prefix script to be used for the Perl scripts in your project. The CodeWarrior IDE treats the prefix script as an implicit `require` file. The `require` directive is the Perl equivalent of the `#include` directive in C and C++. The Perl commands in the prefix file are executed before each Perl script file in your project.

To configure a prefix file:

1. Open the project window for your project.
2. Select the build target you want to configure.

   Note

   You must configure the prefix file separately for each build target in your project.

3. Select ***target_name*** **Settings…** from the **Edit** menu and select Perl Panel from the list of Target Settings Panels. The CodeWarrior IDE displays the File Mappings panel (Figure A-2 on page A-5).

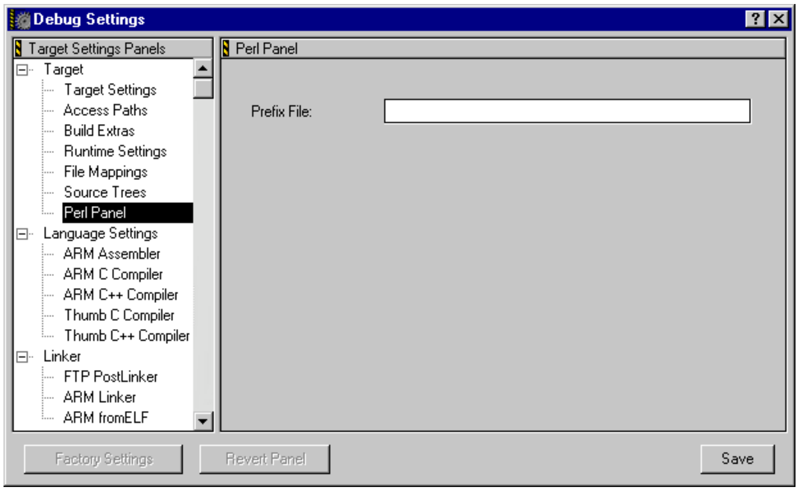


Figure A-2 Perl configuration panel

4. Enter the name of the Perl file you want to use as a prefix file.

   ——— **Note** ———

   The Perl plug-in for CodeWarrior uses the find-and-load functionality of the CodeWarrior IDE. This functionality depends on the ability of the IDE to find referenced files using absolute paths. You must specify the access paths for any files in the Perl script or the Prefix file that are not referenced by absolute paths. See *Configuring access paths* on page 9-20 for more information.

   ————————

5. Click **Save** to save your changes. The CodeWarrior IDE will execute Perl commands in the prefix file prior to executing each Perl file in the project.

A.3 Using Perl scripting

This section describes how to add Perl files to your project, and gives an example Perl script.

A.3.1 Adding Perl files to you project

To add a Perl file to your project:

1. Ensure that you have installed the Perl plug-in software and have configured the CodeWarrior IDE to recognize the filename extension for your Perl files. See *Perl software plug-ins* on page A-2 and *Configuring your project for Perl* on page A-3 for more information.

2. Select **Add Files…** from the **Project** menu. The CodeWarrior IDE displays the Add Files dialog box (Figure A-3).

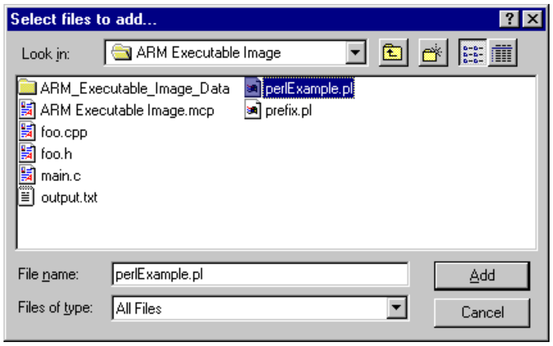


Figure A-3 Adding Perl source to a project

3. Select **All Files** from the Files of Type pop-up menu to display `.pl` files in the dialog.

4. Select the Perl file you require and click **Add**. The CodeWarrior IDE adds the Perl file to your project.

Changing the build order of added files

The CodeWarrior IDE executes Perl scripts in the order in which they appear in the Link Order view, regardless of their order in the Files view. For example, if a Perl script is listed after a C source file in the Link Order view, the CodeWarrior IDE calls the ARM C compiler to compile the source file, and then calls the MW Perl plug-in to process the Perl script.

To change the order in which source files and Perl scripts are processed, use drag and drop in the Link Order view. See *Setting the link order* on page 2-78 for more information.

A.3.2 Restrictions

The following usage restrictions and special considerations apply to using Perl with the CodeWarrior IDE.

StdIn Usage

`StdIn` is not supported in the prefix file. This means that the Perl script cannot accept keyboard input.

A.3.3 Example

Example A-1 shows a simple example of a Perl script.

Example A-1

```
# Simple Perl Example

# Print a line of text
print "Hello World!\n";

$scale0 = 0;
$scale1 = 1;
$scale2 = 2.5;

# Create and open a file for output
open (theFile, ">output.txt");

# Dump some text into the file
print theFile "The file should now be open\n";
print theFile "Let's try a few things:\n\n";

# Arithmetic
print theFile "**Arithmetic \n";
print theFile $scale1 + $scale2 . "\n";
print theFile $scale1 * $scale2 . "\n";
print theFile $scale1 % $scale2 . "\n\n";

# Boolean logic
print theFile "**Boolean \n";
print theFile ($scale0 && $scale0) . "\n";
print theFile ($scale0 && $scale1) . "\n";
print theFile ($scale1 && $scale1) . "\n";

print theFile ($scale0 || $scale0) . "\n";
print theFile ($scale0 || $scale1) . "\n";

print theFile (!$scale0) . "\n\n";

# Comparison
print theFile "**Comparisons \n";
print theFile ($scale2 == $scale2) . "\n";

print "That's it, closing file\n";

# Close the file
close theFile;
```

Appendix B
CodeWarrior 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 B-2
- *CodeWarrior IDE default key bindings* on page B-25.

B.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 B-3
- *Edit menu* on page B-5
- *Search menu* on page B-8
- *Project menu* on page B-12
- *Browser menu* on page B-17
- *Window menu* on page B-18
- *Version Control System (VCS) menu* on page B-21
- *Help menu* on page B-22
- *Toolbar submenu* on page B-23.

Table B-1 summarizes 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.

Table B-1 Menu command availability

Menus always available	Context-sensitive menus
File	**Data** (This menu is not used by CodeWarrior for the ARM Developer Suite.)
Edit	**Browser**
Search	**Catalog** (This menu is not used by CodeWarrior for the ARM Developer Suite.)
Project	**Layout** (This menu is not used by CodeWarrior for the ARM Developer Suite.)
Debug (This menu is not used by CodeWarrior for the ARM Developer Suite.)	
Window	
Help	

The **Version Control System (VCS)** menu is displayed only if you have installed and configured the CodeWarrior IDE to work with a compatible revision control system that you purchased separately. See the documentation that came with the additional revision control software for more information on revision control systems, and how to use them with the CodeWarrior IDE.

B.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.

New Text File

This command creates a new editable text file. See *Creating a new file* on page 4-3 for more information.

New…

This command opens the New dialog box. This dialog box helps you 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 4-5 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 in order to distinguish them. See *Opening files from the File menu* on page 4-5 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 4-9 for more information.

Find and Open ‘Filename’

This command opens an existing text file, using the currently selected text in the editor window as the target file name. See *Opening header files from an editor window* on page 4-9 for more information.

Close

This command closes the active window. See *Closing files* on page 4-16 for more information.

Close All

This command closes all open windows of a certain type. The menu command is based on the type. For example, when several editor windows are currently open and one of them is selected, the menu command is **Close All Editor Documents**. See *Closing all files* on page 4-17 for more information.

Save

This command saves the contents of the active window to disk. See *Saving editor files* on page 4-12 for more information.

Save All

This command saves all editor files that are currently open. See *Saving all files* on page 4-13 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 4-13 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 4-14 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 4-20 for more information.

Import Components...

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Close Catalog

This menu item is not used by CodeWarrior for the ARM Developer Suite.

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 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 2-23 for more information.

Export Project...

This command exports a CodeWarrior 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 2-23 for more information.

Print Setup...

This command sets the options used when printing files from the CodeWarrior IDE. See *Setting print options* on page 4-18 for more information.

Print...

This command prints files from the CodeWarrior IDE on your printer. See *Printing a window* on page 4-18 or read the documentation that accompanies your printer 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.

B.1.2 Edit menu

The **Edit** menu contains all the customary editing commands, along with some CodeWarrior additions. This menu also includes the commands that open the Preferences and Target Settings windows.

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 will remove the text you have just typed.

See *Undoing the last edit* on page 5-15 and *Undoing and redoing multiple edits* on page 5-15 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 8-15 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 with regard 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 5-15 and *Undoing and redoing multiple edits* on page 5-15 for more information on undo and redo operations. See *Editor settings* on page 8-15 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 dimmed 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.

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 5-12 for more information.

Balance

This command selects the text enclosed in either parentheses (), brackets [], or braces {}. For a complete procedure on how to use this command, and how to balance while typing, see *Balancing punctuation* on page 5-14.

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 5-15 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 5-15 for more information.

Preferences...

Use this command to change the global preferences for the CodeWarrior IDE. See *Choosing general preferences* on page 8-6 for more information.

Targetname Settings

Use this command to display the Target Settings window where you can change settings for the active build target. The name of this menu command will vary depending on the name of your current build target.

See *Configuring target settings* on page 9-14 for more information on the Settings window. See *Set Current Target* on page B-15 for information on changing the current build target.

Version Control Settings...

This menu command displays the **Version Control System** options panel. See Chapter 10 *Using CodeWarrior IDE with Version Control Systems* for more information.

If this command is not enabled, you do not have a revision control system configured for use with the CodeWarrior IDE.

Commands & Keybindings...

Use this menu item to set keybinding for commands, and to customize the CodeWarrior IDE toolbars.

B.1.3 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.

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 6 *Searching and Replacing Text* for more information.

Find Next

This command finds the next occurrence of the **Find text box** string in the active window. This is an alternative to clicking the **Find** button in the **Find** dialog box. See *Finding and replacing text with the Find dialog* on page 6-4 for more information.

Find Previous

Find Previous operates the same way as **Find Next**, except that it finds the previous occurrence of the **Find text box** string. See *Finding and replacing text with the Find dialog* on page 6-4 for more information.

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 dimmed. See *Finding and replacing text in multiple files* on page 6-8 for more information.

Find in Previous File

This command operates in much the same way as **Find in Next File**. The **Find in Previous File** command begins at the end of the previous file in the file list and searches for the next occurrence of the **Find text box** string. See *Finding and replacing text in multiple files* on page 6-8 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 5-12 for more information.

Enter Replace String

This command copies the selected text in the active window into the **Replace text box**, making it the replacement string. This is an alternative to selecting the string and copying it into the Find window. See *Finding and replacing text with the Find dialog* on page 6-4 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 dialog* on page 6-4 for more information.

Find Previous Selection

This command finds the previous occurrence of the selected text in the active text editor window. See *Finding and replacing text with the Find dialog* on page 6-4 for more information. See *Selecting text* on page 5-12 for more information on how to select text.

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 dimmed.

This command is useful if you want to replace one instance of a text string without having to open the Find window. For example, say that you have just replaced all the occurrences of the variable `icount` with `jcount`. While scrolling through your source code, you notice one instance of the variable `icount` is misspelled as `icont`. To replace this variable with `jcount`, select `icont` and select **Replace** from the **Search** menu. See *Finding and replacing text with the Find dialog* on page 6-4 for more information. See *Selecting text* on page 5-12 for more information on selecting text.

Replace & 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 dimmed. See *Finding and replacing text with the Find dialog* on page 6-4 for more information. See *Selecting text* on page 5-12 for more information on selecting text.

Replace & Find Previous

This command operates the same way as **Replace & Find Next**, except that it performs a **Find Previous** after replacing 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 dimmed.

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, consult *Using the message window* on page 3-15.

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 7-22 for more information.

Go Forward

This command moves you to the next view in the browser (after you have used the **Go Back** command to return to a previous view). See *Using Go Back and Go Forward* on page 7-22 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 5-20 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 4-21 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.

B.1.4 Project menu

The **Project** menu lets you add and remove files and libraries from your project. It also lets you compile, build, and link your project. All of these commands are discussed in this section.

Add Window

This command adds the file in the active editor window to the open project. See *Adding the current editor window* on page 2-42 for more information.

Add Files...

This command adds files to the project window. See *Using the Add Files command* on page 2-39 for more information.

Create New Group...

The **Create New 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 2-42 for more information.

Create New Target

The **Create New 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 2-53 for more information.

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 dimmed.

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 3-17.

Preprocess

This command performs preprocessing on selected C and C++ source code files. See *Preprocessing source code* on page 2-77 for more information.

Precompile

This menu option is not used by CodeWarrior for the ARM Developer Suite.

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 2-72 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 2-81 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 2-76 for more information.

Make

This command builds the selected project by compiling and linking the modified and touched files in the open project. The results of a successful build depend on the selected project type. See *Making a project* on page 2-77 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 2-79 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. **Re-search for files** 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 will find the source files in the same location (the exception is if the file no longer exists in the old location). In this case the CodeWarrior IDE will only re-search for header files. To force the CodeWarrior IDE to also re-search for source files, you must first select **Reset Project Entry Paths**.

If the **Save Project Entries Using Relative Paths** setting is disabled, the CodeWarrior IDE will re-search 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 will re-search 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 2-48 for more information.

Enable Debugger and Disable Debugger

Use these commands to turn debugging on or off for the current build target. The command displayed depends on the current debug status of the target. See *Controlling debugging in a project* on page 3-4 for more information.

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 dimmed.

Debug

This command compiles and links the current build target, and launches an ARM debugger to debug the output image.

Set Default Project

This menu command selects which project is the default project. See *Choosing a default project* on page 2-22 for more information.

Set Current Target

This command enables you to choose a different target within the current project to work with. This menu command might be useful if you want to switch between multiple targets in a project and do a build for each one.

B.1.5 Debug menu

The Debug menu is not used by CodeWarrior for the ARM Developer suite. For more information on debugging your code using the ARM debuggers see:

- Chapter 3 *Working with the ARM Debuggers*
- *ADS Debuggers Guide*.

The following menu items are documented for completeness only.

Kill

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Restart

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Step Over

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Step Into

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Step Out

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Stop

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Set Breakpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Clear Breakpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Enable Breakpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Disable Breakpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Clear All Breakpoints

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Show Breakpoints

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Hide Breakpoints

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Set Watchpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Clear Watchpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite

Enable Watchpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Disable Watchpoint

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Clear All Watchpoints

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Break on C++ Exception

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Break on Java Exceptions

This menu item is not used by CodeWarrior for the ARM Developer Suite.

Switch to Monitor

This menu item is not used by CodeWarrior for the ARM Developer Suite.

B.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.

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…

This menu item is not used by CodeWarrior for the ARM Developer Suite.

New Method…

This menu item is not used by CodeWarrior for the ARM Developer Suite.

New Event Set…

This menu item is not used by CodeWarrior for the ARM Developer Suite.

New Event…

This menu item is not used by CodeWarrior for the ARM Developer Suite.

B.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.

Stack Editor Windows

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 dimmed when the active window is the project window or messages window.

Tile Editor Windows

This command arranges all editor windows so that none overlap. This command is dimmed when the active window is the project window or messages window.

Tile Editor Windows Vertically

This command arranges all the editor windows in a single row.

This command is disabled when the active window is the project window or messages window.

Zoom Window

This menu command expands the active window to the largest possible size. If you select the menu command again, the window returns to its original size.

Minimize Window

This command minimizes the currently selected window.

Restore Window

This command restores the currently selected minimized window to its original size.

Save Default Window

This command saves the settings of the active browser window, so that the next time you open a browser window, the CodeWarrior IDE opens it with the saved settings. See *Saving editor window settings* on page 5-9 or *Saving a default Class browser window* on page 7-16 for more information.

Toolbar

This command causes the **Toolbar** submenu to appear. See *Toolbar submenu* on page B-23 for more information.

Browser Contents

This command displays the browser Contents window. This menu command is dimmed when the browser is not activated. See *Viewing data by type with the Contents view* on page 7-17 for more information. See *Activating the browser* on page 7-5 for details of how to activate the browser.

Class Hierarchy Window

This command displays the browser Multi-Class Hierarchy window. This menu command is dimmed when the browser is not activated. See *Multi-class hierarchy window* on page 7-18 for more information. See *Activating the browser* on page 7-5 for details of how to activate the browser.

New Class Browser

This command displays the browser Class window. This menu command is dimmed when the browser is not activated. See *Viewing data by class with the Class browser view* on page 7-8 for more information. See *Activating the browser* on page 7-5 for details of how to activate the browser.

Build Progress Window

This command displays the progress window for builds, as shown in Figure 2-54 on page 2-75.

Errors & Warnings Window

This command displays the Errors and Warnings window. See *Using the message window* on page 3-12 for more information. See also *Using batch searches* on page 6-6.

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 2-4 for more information.

Processes Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

Expressions Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

Global Variables Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

Breakpoints Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

Watchpoints Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

Register Window

This window is not used by CodeWarrior for the ARM Developer Suite. See Chapter 3 *Working with the ARM Debuggers* for more information.

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 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.

B.1.8 Version Control System (VCS) menu

The Version Control System (VCS) menu, similar to that shown in Figure B-1 on page B-22, is displayed in the menu bar of the CodeWarrior IDE if you are using a Version control system.

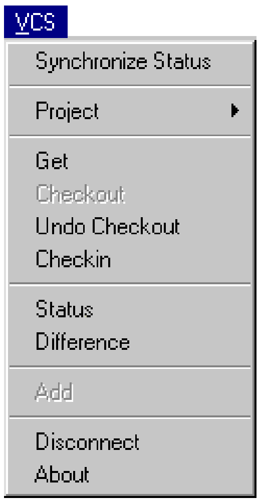


Figure B-1 VCS menu

B.1.9 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.

CodeWarrior Help

This menu item is not used by CodeWarrior for the ARM Developer Suite.

How to...

This command opens the main help file for CodeWarrior for the ARM Developer Suite.

Glossary

This command displays the CodeWarrior IDE glossary of terms.

IDE

This command opens the main help file for CodeWarrior for the ARM Developer Suite.

Debugger

This command displays the online help that describes how the ARM debuggers interact with CodeWarrior for the ARM Developer Suite.

Error Reference

This menu option is not used by CodeWarrior for the ARM Developer Suite.

C/C++ Compiler Reference

This menu option is not used by CodeWarrior for the ARM Developer Suite. Refer to the ADS online books for information on the ARM C and C++ compilers.

MSL C Reference

This menu option is not used by CodeWarrior for the ARM Developer Suite. Refer to the ADS online books for information on the ARM C libraries.

MSL C++ Reference

This menu option is not used by CodeWarrior for the ARM Developer Suite. Refer to the ADS online books for information on the ARM-supplied Rogue Wave C++ libraries.

Other

This menu option is not used by CodeWarrior for the ARM Developer Suite.

About Metrowerks

This command displays the Metrowerks About Box.

B.1.10 Toolbar submenu

The **Window** menu has another 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 8-37 for more information.

Show Window Toolbar

This command cause the CodeWarrior IDE to display the toolbar in the active window. The actual command shown in the menu will toggle between **Show Window Toolbar** and **Hide Window Toolbar**, depending on whether the active window's toolbar is visible.

Hide Window Toolbar

This command cause the CodeWarrior IDE to hide the toolbar in the active window. The actual command shown in the menu will toggle between **Show Window Toolbar** and **Hide Window Toolbar**, depending on whether the active window's toolbar is visible.

Reset Window Toolbar

This command causes the toolbar in the active window to reset to a default state. You should 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. Once 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 Main Toolbar

This command displays the main window toolbar.

Hide Main Toolbar

This command hides the main window toolbar.

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.

B.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 8-26.

The key bindings sections include:

- *File menu* on page B-25
- *Edit menu* on page B-26
- *Search menu* on page B-27
- *Project menu* on page B-28
- *Window menu* on page B-29
- *Miscellaneous* on page B-30
- *Editor commands* on page B-31.

B.2.1 File menu

Table B-2 lists the default key bindings for manipulating projects and files from within the CodeWarrior IDE.

Table B-2 File key bindings

Command	Key binding
New	Ctrl-N
New…	Ctrl-Shift-N
Open	Ctrl-O
Find and Open ‘selection’	Ctrl-D
Find and Open File	Ctrl-Shift-D
Close	Ctrl-W
Close All	Ctrl-Shift-W
Save	Ctrl-S
Save All	Ctrl-Shift-S
Save As	-

Table B-2 File key bindings (continued)

Command	Key binding
Save A Copy As	-
Revert	-
Page Setup	-
Print	Ctrl-P
Quit	-

B.2.2 Edit menu

Table B-3 contains the default key bindings for the commands in the **Edit** menu of the CodeWarrior IDE.

Table B-3 Edit key bindings

Command	Key binding
Undo	Ctrl-Z
Redo	Ctrl-Shift-Z
Cut	Ctrl-X
Copy	Ctrl-C
Paste	Ctrl-V
Clear	-
Select All	Ctrl-A
Balance	Ctrl-B
Shift Left	Ctrl-[
Shift Right	Ctrl-]
Insert Reference Template	-
Preferences	-
Target Settings	Alt-F7
VCS Settings	-

B.2.3 Search menu

Table B-4 contains the default key bindings for the commands in the **Search** menu of the CodeWarrior IDE.

Table B-4 Search key bindings

Command	Key binding
Find	Ctrl-F
Find Next	F3
Find Previous	Shift-F3
Find in Next File	Ctrl-T
Find in Previous File	Ctrl-Shift-T
Enter Find String	Ctrl-E
Enter Replace String	Ctrl-Shift-E
Find Selection	Ctrl-F3
Find Previous Selection	Ctrl-Shift-F3
Replace	Ctrl-=
Replace & Find Next	Ctrl-L
Replace & Find Previous	Ctrl-Shift-L
Replace All	-
Find Definition	Ctrl-’
Find Definition & Reference	-
Find Reference	-
Go Back	Ctrl-Shift-B
Go Forward	Ctrl-Shift-F
Goto Line	Ctrl-G
Compare Files	-
Apply Difference	-
Unapply Difference	-

B.2.4 Project menu

Table B-5 contains the default key bindings for commands in the **Project** menu.

Table B-5 Project key bindings

Command	Key binding
Add Window	-
Add Files	-
Create Group/ Segment/Target	-
Check Syntax	Ctrl-;
Preprocess	-
Precompile	-
Compile	Ctrl-F7
Disassemble	-
Bring Up To Date	Ctrl-U
Make	F7
Stop Build	Ctrl-Break
Remove Object Code	Ctrl - –
Re-search For Files	-
Reset Project Entry Paths	-
Synchronize Modification Dates	-
Enable Debugging	-
Run/Debug	F5
Debug/Run	Ctrl-F5

B.2.5 Window menu

Table B-6 contains the default key bindings for handling many common windows in the CodeWarrior IDE.

Table B-6 Window menu key bindings

Command	Key binding
Stack	-
Tile	-
Tile Vertical	-
Zoom Window	Ctrl-/
Collapse/Expand Window	-
Save Default Window	-
Browser Catalog Window	-
Class Hierarchy Window	-
New Class Browser	Alt-F12
Build Progress Window	-
Errors & Warnings Window	Ctrl-I
Project Inspector	Alt-Enter
ToolServer Worksheet	-
Processes Window	-
Expressions Window	Alt-Shift-3
Global Variables Window	-
Breakpoints Window	Alt-F9
Watchpoints Window	-
Select Default Project	Ctrl-0
Select Document 1	Ctrl-1
Select Document 2	Ctrl-2
Select Document 3	Ctrl-3

Table B-6 Window menu key bindings (continued)

Command	Key binding
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

B.2.6 Miscellaneous

Table B-7 contains the default key bindings for handling miscellaneous tasks in the CodeWarrior IDE.

Table B-7 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
Run Script	-
Stop Script	-

B.2.7 Editor commands

Table B-8 contains the default key bindings for handling editor windows in the CodeWarrior IDE.

Table B-8 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 B-8 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-,

Appendix C
CodeWarrior IDE Installation and Preference Settings

This appendix describes how to install multiple copies of the CodeWarrior IDE, and how to use CodeWarrior for the ARM Developer Suite with other versions of the CodeWarrior IDE. It contains the following sections:

- *The CodeWarrior preferences directory* on page C-2
- *Using different versions of the CodeWarrior IDE* on page C-3.

C.1 The CodeWarrior preferences directory

The CodeWarrior IDE maintains preferences and persistence information in the following locations:

For Windows NT `c:\Winnt\Metrowerks\CodeWarrior Pro 5 IDE Prefs`

For Windows 95/98 `c:\Windows\Metrowerks\CodeWarrior Pro 5 IDE 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 CodeWarrior for the ARM Developer Suite you can preconfigure a single installation and copy the preferences directory for that installation to each machine.

CodeWarrior preferences are deleted when you uninstall the CodeWarrior for the ARM Developer Suite.

C.2 Using different versions of the CodeWarrior IDE

CodeWarrior for the ARM Developer Suite is customized to support the ARM tool chain. This means that:

- your CodeWarrior IDE preferences might not be applicable to other CodeWarrior versions
- some components of the CodeWarrior IDE that are registered in the Windows registry at installation are specific to CodeWarrior for the ARM Developer Suite.

To switch from CodeWarrior for the ARM Developer Suite to another version of the CodeWarrior IDE on the same machine:

1. Rename the CodeWarrior 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 CodeWarrior for the ARM Developer Suite, rename your preferences directories and run `regservers.bat` from *`install_directory`*`\ARM\ADSv1_1\Bin`.

Glossary

ADS	See *ARM Developer Suite*.
ADU	See *ARM Debugger for UNIX*.
ADW	See *ARM Debugger for Windows*.
ANSI	American National Standards Institute. An organization that specifies standards for, among other things, computer software.
ARM Debugger for UNIX	ARM Debugger for UNIX (ADU) is the UNIX version of the ARM Debugger for Windows. This debugger will not be supported in future versions of the ARM Developer Suite.
ARM Debugger for Windows	ARM Debugger for Windows (ADW). This debugger will not be supported in future versions of the ARM Developer Suite.
ARM 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.
ARM eXtended Debugger	The ARM eXtended Debugger (AXD) is the latest debugger software from ARM that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target. AXD is supplied in both Windows and UNIX versions.
ARMulator	ARMulator is an instruction set simulator. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors.

armsd	The ARM Symbolic Debugger (armsd) is an interactive source-level debugger providing high-level debugging support for languages such as C, and low-level support for assembly language. It is a command-line debugger that runs on all supported platforms.
ATPCS	ARM and Thumb Procedure Call Standard, defines how registers and the stack will be used for subroutine calls.
AXD	See *ARM eXtended Debugger*.
Big-Endian	Memory organization where the least significant byte of a word is at a higher address than the most significant byte.
Coprocessor	An additional processor which is used for certain operations. Usually used for floating-point math calculations, signal processing, or memory management.
Debugger	An application that monitors and controls the execution of a second application. Usually used to find errors in the application program flow.
Double word	A 64-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.
DWARF	Debug With Arbitrary Record Format
EC++	A variant of C++ designed to be used for embedded applications.
ELF	Executable and linking format
Environment	The actual hardware and operating system that an application will run on.
Executable and linking format	The industry standard binary file format used by the ARM Developer Suite. ELF object format is produced by the ARM object producing tools such as armcc and armasm. The ARM 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.
Flash memory	Non-volatile memory that is often used to hold application code.
Halfword	A 16-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.
Heap	The portion of computer memory that can be used for creating new variables.
Host	A computer which provides data and other services to another computer.
ICE	In Circuit Emulator.

IDE	Integrated Development Environment (CodeWarrior).
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. *See also* Output sections
Input section	Contains code or initialized data or describes a fragment of memory that must be set to zero before the application starts. *See also* Output sections
Interworking	Producing an application that uses both ARM and Thumb code.
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 *Big-endian*.
Local	An object that is only accessible to the subroutine that created it.
Load view	The address of regions and sections when the image has been loaded into memory but has not yet started execution.
Memory management unit	Hardware that controls caches and access permissions to blocks of memory, and translates virtual to physical addresses.
MMU	See *Memory Management Unit*.
Multi-ICE	Multi-processor in-circuit emulator. ARM registered trademark.
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
PCS	Procedure Call Standard. *See also* ATPCS

PIC	Position Independent Code. *See also* ROPI
PID	Position Independent Data *or* the ARM Platform-Independent Development card. *See also* RWPI
Reentrancy	The ability of a subroutine to have more that one instance of the code active. Each instance of the subroutine call has its own copy of any required static data.
Regions	In an Image, a region is a contiguous sequence of one to three output sections (RO, RW, and ZI).
Retargeting	The process of moving code designed for one execution environment to a new execution environment.
RO	Read-write.
RW	Read-only.
ROPI	Read Only Position Independent. Code and read-only data addresses can be changed at run-time.
RWPI	Read Write Position Independent. Read/write data addresses can be changed at run-time.
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.
SWI	Software Interrupt. An instruction that causes the processor to call a programer-specified subroutine. Used by ARM to handle semihosting.
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.
Veneer	A small block of code used with subroutine calls when there is a requirement to change processor state or branch to an address that cannot be reached in the current processor state.
VCS	Version Control System.
Watchpoint	A location within the image which will be monitored and which will cause execution to break when it changes.
Word	A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.
ZI	Zero-initialized.

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.