RealView® Developer Kit

Version 2.2

Project Management User Guide

ARM®

RealView Developer Kit
Project Management User Guide

Copyright © 2005, 2006 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

Proprietary Notice

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

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

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

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

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com

Contents
RealView Developer Kit Project Management User Guide

Glossary

Preface

This preface introduces the *RealView® Developer Kit (RVDK) v2.2 Project Management Guide* that shows you how to use RealView® Debugger to manage software projects. It contains the following sections:

About this book

This book describes how to use RealView Debugger to manage software projects and work with images:

- an explanation of the features of the RealView Debugger so that you can manage your software projects and organize source files

- examples describing how to configure project properties

- a glossary of terms for users new to RealView Debugger.

Intended audience

This book is written for developers who are using RealView Debugger to manage ARM®-targeted development projects. It assumes that you are an experienced software developer, and that you are familiar with the ARM development tools. It does not assume that you are familiar with RealView Debugger.

Examples

Your hardware and software might not be the same as those used for testing the examples given in this book, so it is possible that certain addresses or values might vary slightly from those shown, and some of the examples might not apply to you. In these cases you might have to modify the instructions to suit your own circumstances.

The examples in this book use the ARM-targeted programs stored in
`install_directory`\RVDK\Examples\... .

Using this book

This book is organized into the following chapters:

Chapter 1 *Overview of Project Management*

Read this chapter for an introduction to projects in RealView Debugger. It provides an overview of the concepts underlying project management and introduces the facilities that you can use to manage your development projects.

Chapter 2 *Working with Projects*

Read this chapter for details on opening projects in RealView Debugger. It describes how to use the **Project** menu and introduces some of the tasks you might have to complete when developing software projects.

You can use the programs stored in
install_directory\RVDK\Examples\... to try the options described in this
chapter.

Chapter 3 *Creating New Projects*

Read this chapter for details on creating your own projects in RealView
Debugger.

You can use the programs stored in
install_directory\RVDK\Examples\... to try the options described in this
chapter.

Chapter 4 *Using the Project Properties Window*

This chapter describes how to view project properties and how to change
settings to begin customizing your projects.

Chapter 5 *Customizing Projects*

This chapter describes how to change project properties and includes
examples illustrating how to customize your projects.

Chapter 6 *Managing Projects in the Code Window*

This chapter explains how to use the RealView Debugger GUI to manage
your projects. It describes how to change the project environment using
the Project Control dialog box and how to access project files in the
Process Control pane.

Chapter 7 *Building Applications*

Read this chapter for details on the build process and how to customize it.

Read this chapter for details on the **Tools** menu that enables you to build
your applications. It is recommended that you read this chapter when
creating your own projects.

Chapter 8 *Managing Build Target Configurations*

Read this chapter for details on how to manage build target configurations
to define how development projects are built.

Chapter 9 *Project Binding*

Read this chapter for details on how RealView Debugger *binds* projects
to connections. It also describes how this binding mechanism can be used
to load the project image automatically and execute RealView Debugger
commands.

Chapter 10 *Working with Multiple Projects*

> Read this chapter for details on how to work with multiple projects in a debugging session.

Chapter 11 *Editing Source Code*

> Read this chapter for a description of the file editor that is supplied as part of RealView Debugger.

Chapter 12 *Searching and Replacing Text*

> Read this chapter for a description of using RealView Debugger to search and replace text in source files.

Chapter 13 *Working with Version Control Systems*

> Read this chapter for a description of using the version control options in RealView Debugger.

Glossary   An alphabetically arranged glossary defines the special terms used in this book.

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
<u>mono</u>space	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
monospace italic	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
monospace bold	Denotes language keywords when used outside example code.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information.

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

ARM publications

This book is part of the RealView Debugger documentation suite. Other books in this suite include:

*   *RealView Developer Kit v2.2 Debugger User Guide* (ARM DUI 0281)
*   *RealView Developer Kit v2.2 Compiler and Libraries Guide* (ARM DUI 0282)
*   *RealView Developer Kit v2.2 Assembler Guide* (ARM DUI 0283)
*   *RealView Developer Kit v2.2 Command Line Reference* (ARM DUI 0284)
*   *RealView Developer Kit v2.2 Linker and Utilities Guide* (ARM DUI 0285)

See the datasheet or Technical Reference Manual for information relating to your hardware.

Other publications

For a comprehensive introduction to ARM architecture see:

Steve Furber, *ARM system-on-chip architecture* (2nd edition, 2000). Addison Wesley, ISBN 0-201-67519-6.

For a detailed introduction to regular expressions, as used in the RealView Debugger search and pattern matching tools, see:

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

For the definitive guide to the C programming language, on which the RealView Debugger macro and expression language is based, see:

Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language* (2nd edition, 1989). Prentice-Hall, ISBN 0-13-110362-8.

For more information about the JTAG standard, see:

IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std. 1149.1), available from the IEEE (`http://www.ieee.org`).

Feedback

ARM Limited welcomes feedback on both RealView Debugger and its documentation.

Feedback on RealView Debugger

If you have any problems with RealView Debugger, submit a Software Problem Report:

1. Select **Help** → **Send a Problem Report...** from the RealView Debugger main menu.

2. Complete all sections of the Software Problem Report.

3. To get a rapid and useful response, give:

    • a small standalone sample of code that reproduces the problem, if applicable

    • 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.

4. Email the report to your supplier.

Feedback on this book

If you have any comments on this book, 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 your comments.

General suggestions for additions and improvements are welcome.

 ARM DUI 0291C

Chapter 1
Overview of Project Management

This chapter gives an overview of projects in RealView® Debugger. It contains the following sections:

- *What is a project?* on page 1-2
- *User-defined projects* on page 1-4
- *Auto-projects* on page 1-6
- *Organizing user-defined projects* on page 1-7
- *Project properties* on page 1-9
- *Build tools for user-defined projects* on page 1-10
- *Build target configurations* on page 1-11
- *Makefiles* on page 1-12
- *Project binding* on page 1-14
- *Projects in the workspace* on page 1-15
- *Projects in Connection Properties* on page 1-16.

1.1   What is a project?

A project is the highest level structural element that you can use to organize your source files and determine their output. The project information and settings can be managed by RealView Debugger.

Setting up a project in RealView Debugger is not required for debugging, but it can provide an aid to development. RealView Debugger uses projects to save your list of files, understand your build model, and maintain a record of your project-level preferences. A project also enables RealView Debugger to save, and load automatically, specified debugging states, for example breakpoints.

A user-defined project can also speed up your debugging session because your project stores information and settings about your image that can then be used to help locate errors, and rebuild your source code into an executable image or program.

If you have started RealView Debugger, you can begin to use many features of the debugger, for example editing source code and building projects. However, to begin debugging images you must connect to a suitably configured debug target.

RealView Debugger enables you to:
- create a range of software projects using predefined templates
- access image-related settings through auto-projects
- define build tools to use with all projects or only a subset
- view and change project properties without leaving the Code window
- define different build target configurations
- associate projects with connections or specific target processors
- set up a project environment automatically when the workspace opens
- open projects automatically when you connect to a specified debug target.

1.1.1 Types of project

There are two types of project in RealView Debugger:

User-defined projects

Projects that you create and set up. See *User-defined projects* on page 1-4 for more details.

———— **Note** ————

If you have a user-defined project, do not load the image without first opening the project. If you do this, RealView Debugger creates an auto-project and does not load the settings from your project settings file.

————————————

Auto-projects

Projects that RealView Debugger uses automatically when you load an image directly. RealView Debugger defines project properties based on the image contents. An auto-project is a no-build project because RealView Debugger is unable to determine the build model from the image. See *Auto-projects* on page 1-6 for more details.

1.2 User-defined projects

A user-defined project is set up and managed by you. If you have existing source files, RealView Debugger enables you to select the source files during project creation, or add them later.

The project information and settings are stored in a project settings file. This has the same name as the project and an extension of `.prj`. The file is placed in the project directory that you specify when you create a project. For example, `std_proj_1.prj` is the project settings file for the project named `std_proj_1`.

A user-defined project defines:

- a list of source files
- build rules, or specifies no build at all
- build rules including compiler, assembler, and linker
- custom build rules
- build target configurations such as debug, debug/release, and release
- makefiles to use
- runtime settings such as the image name and loading rules
- image-specifics such as semihosting on and vector-catch enabled.

——— **Note** ———

If you have created a user-defined project, it is recommended that you open this first to load and debug the associated image, or images. This avoids the creation of an auto-project and enables you to save any new settings or changes to the build model.

The debugger contains BCD files that describe MCUs and correspondingdevelopment boards. The actual files vary between versions of RVDK.Board files may have semihosting and vector catch on by default.This section describes how to turn off these features when runningcode from Flash memory.

1.2.1 Types of user-defined project

The types of user-defined project you can create and manage in RealView Debugger are:

Standard   A Standard project is composed of sources to compile, and/or assemble, and link. A Standard project might also contain a custom build model. RealView Debugger creates the project makefile automatically.

Library   A Library project enables you to compile and/or assemble files to put into a library. RealView Debugger creates the project makefile automatically.

Custom   A Custom project enables you to specify your own makefile, or to specify an external build program or script, or to use no build model.

Container   A Container project contains other projects and can be used to:

- share components within, and between, development teams
- divide up complex builds into libraries
- contain projects for different processors.

———— **Note** ————

By default, if you open a Container project, you are working on multiple projects. You can perform additional operations on the individual projects in a Container project.

Copy   You can copy existing user-defined projects to try variations of your application program or to test different development environments.

———— **Note** ————

Regardless of the type of project you create, project files can be controlled by your version control software if required. See Chapter 13 *Working with Version Control Systems* for details on how to do this.

1.3   Auto-projects

When an image, for example `test_image.axf`, is loaded to a debug target, RealView Debugger checks to see if an auto-project with the same name (`test_image.axf.apr`) exists in the same location. If an auto-project exists, RealView Debugger opens this to provide selected project settings for the current session. If no auto-project exists, RealView Debugger creates an in-memory auto-project to use in this session by reading some settings directly from the image.

——— **Note** ———

When RealView Debugger creates an auto-project, it derives a project name from the name of the associated image. It is this project name, not the image name, that appears in the title bar of the default Code window. However, only the image name is visible in the Process Control pane.

Auto-projects are used to store configuration information for the image, including:

- semihosting
- vector catching
- automatic breakpoint setting
- start-up and load commands.

The `.apr` file has an identical format to that of a user-defined Custom project settings file. There is no build model for the current image because this was unknown when the image was loaded. The **Process** tab in the Process Control pane contains a `Settings` group that indicates whether or not you have saved the auto-project settings to a `.apr` file.

——— **Note** ———

You can merge the settings from an auto-project when you create a new user-defined project. See *Merging auto-project settings into a project* on page 3-16 for details on how to do this.

See *Working with auto-projects* on page 6-7 for details on how to manage auto-projects.

1.4 Organizing user-defined projects

A user-defined project is a collection of source files, library files, and other input files. You can organize the files in a project in different ways to provide a logical structure to your source.

It is recommended that you decide the best way to organize your user-defined project files before using the RealView Debugger project management options. How projects are organized affects the extent to which files can be shared between developers.

When you create a user-defined project, you specify the project name and the *project base directory* (see *Creating a new user-defined project* on page 3-2). The source files used in the project do not have to be in the project base directory but can be located elsewhere and are referred to using relative pathnames, where possible.

RealView Debugger generates a warning message when projects are not self-contained so that you can decide to cancel an operation or continue. In general, keeping source files together within the project base directory, and any subdirectories, is the preferred option for single-user projects.

When working with projects and making changes, additional files are created, for example safety backup files.

1.4.1 Project size

There is no limit to the number of files that RealView Debugger can handle in a single project. However, your operating system might impose a limit on the number of files that can be passed to the linker. Keeping all your source files in the project base directory, and using short filenames, can help to maximize the number of files in a project. However, using libraries is the recommended approach for large projects.

There is a limit imposed on the line length in the generated makefile. This is defined by the setting `MAXLINELENGTH`, in the file `\etc\startup.mk`. Set to 32768, you can make this longer by editing the file.

1.4.2 Deleting user-defined projects

In general, you do not have to delete user-defined project files or the contents of a project base directory. After you have created a user-defined project, you can add and delete files as necessary using the project management options from the Code window. You can also change your build model and other project components using the Project Properties window.

Deleting user-defined project files is not recommended where projects are not single-user, self-contained projects as this might prevent other developers from accessing your source files, built files, or build model.

1.5 Project properties

The Project Properties window enables you to view the project settings for user-defined projects or auto-projects. You can use this to change user-defined project options as build configurations change, or to build an executable image without having to manually edit any files used by RealView Debugger. You are recommended to use this interface because this populates the interrelated files that control the build process and constructs the required makefiles.

However, you can choose not to use this interface and to set up your own commands and scripts if required.

If you are using an auto-project, you can use this interface to define image load options that are stored with the current image and used the next time the image is loaded.

For details on how to manage and configure project properties see:

- Chapter 4 *Using the Project Properties Window*
- Chapter 5 *Customizing Projects*.

1.6 Build tools for user-defined projects

RealView Debugger provides support for multiple build tools in user-defined projects. The project *toolchain* defines the base settings for development tools, for example *RealView Developer Kit* (RVDK). When you have specified these settings they are used for every project that you build using that toolchain. You can override these settings in your project so that different tools are used for a project or for a particular set of files forming part of a project.

Depending on your installation, RealView Debugger can locate toolchains for projects using environment variables already set, for example RVDK.

It is recommended that you define the location of your build tools before creating your first project. If you do not do this, RealView Debugger asks you to specify it when saving your first project. See *Managing your build tools* on page 7-5 for full details.

1.7 Build target configurations

The most important element in a user-defined project is the *build target configuration*. This defines how the source files within your project are processed, not the project itself, and enables you to build the same image in different ways. A build target configuration is, therefore, a specific arrangement 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.

——— **Note** ———

A build target configuration is distinct from a debug target, such as an ARM® development board.

A user-defined project defines at least one build target configuration, for example a `Debug` build, or a `Release` build. RealView Debugger defines three build target configurations:

Debug   This builds output files that you can fully debug, at the expense of optimization.

DebugRel   This builds output files that provide adequate optimization and give a good debug view.

Release   This builds output files that are fully optimized, at the expense of debug information.

You can define a specific build order for the build target configurations in a project. Build target configurations can share files in the same project, while using their own build settings. The Project Properties window enables you to define and set up such relationships.

Each build target configuration has a corresponding directory where the built files are placed. The directories have the same name as the build target configuration, and are subdirectories of your top-level project directory.

You can create your own build target configurations, and assign different customized settings to each configuration. For example, you can assign different compilation controls for the ARM C compiler to each configuration. For detailed instructions on managing your build target configurations, see Chapter 8 *Managing Build Target Configurations*.

1.8   Makefiles

Makefiles are used to describe the relationships between files in your application program and provide commands for building these files. They provide a database of rules detailing information about files and specifying how they are used in a build. Makefiles are used, therefore, to automate the build process. Projects can wrap user-defined makefiles, or you can edit the RealView Debugger templates to customize your build process, for example to use specific source control.

1.8.1   Standard and Library project makefiles

Creating a Standard or Library project means that makefiles are built for you using the RealView Debugger templates.

When you configure and save your project settings file, RealView Debugger uses a template file named gen_***.mk to generate the required makefile for your project. Different templates are installed depending on the installation type, that is Custom or Typical, and the licenses you possess.

The template name is chosen based on the toolchain you specify when you create the project, that is the target processor and build tools. For example, the template \etc\gen_arx.mk is used for building executable files and libraries with ARM compilation tools to run on the ARM family of processors. This template provides the RVDK and ADS specific makefile layouts for the genmake utility used by RealView Debugger.

You can copy and edit the template file to add details specific to your requirements. If you do this, you can then specify the new template file in your project BUILD group.

The file \etc\startup.mk is used by the make utility when working with projects on Windows to define default settings.

A makefile is created in your project directory for each build target configuration. The following makefiles are created for the default build target configurations, where *projectname* is the name of your project:

- *projectname*_Debug.mk
- *projectname*_DebugRel.mk
- *projectname*_Release.mk.

1.8.2   Custom project makefiles

Creating a Custom project means that you can use your own makefiles. RealView Debugger does not automatically generate makefiles for a Custom project.

See *Steps for creating a Custom project* on page 3-8 for instructions on creating these projects.

——— **Note** ———

When you create a Custom project, RealView Debugger specifies a default make command, that includes the control character $e. To successfully build your Custom project, remove the $e control character, and use your own arguments as required. See *Using your own make command* on page 3-10.

1.9 Project binding

When you create a user-defined project, you define a toolchain associated with the target application. If you are connected to a target and you open this project, RealView Debugger tries to bind the project to the connection using this *default binding*. If successful, the project name is enclosed in round brackets in the default Code window title bar, for example (dhrystone). The Process Control pane (if visible) is updated to show details of the current process and enables you to access the project properties.

If you open a project and there is no connection to bind to, the project is *unbound*. In this case, the project name is enclosed in angled brackets when it is displayed in the Code window title bar, for example <dhrystone>. Although, it is not visible in the Process Control pane, you can still access the project properties from the **Project** menu.

A connection can only have one project bind to it at any one time. If you open a project with a project already bound to your connection, RealView Debugger asks for confirmation to use default binding to bind the new project.

When a project binds to a connection, selected actions can be carried out on that connection, for example image loading actions, or RealView Debugger can execute commands saved in the project settings file.

When you are working with multiple projects, binding enables you to access your image details, view the project properties, and make changes to the project quickly.

When you load an image directly to a target, the auto-project (associated with the image) binds to the connection by default.

You can change binding manually using the Project Control dialog box, see *Using the Project Control dialog box* on page 6-2 for details.

See Chapter 9 *Project Binding* for full details on project binding.

1.9.1 Autobinding

You can define how RealView Debugger binds a project using *autobinding*. You can specify exactly the processor (or processors) to bind to and so restrict your project to specified debug targets. *Autobound* projects take precedence when RealView Debugger tries to bind, depending on the project environment. See *Autobinding* on page 9-3 for full details.

1.10   Projects in the workspace

RealView Debugger saves a project load list when the current workspace closes. This is a list of open projects maintained when the debugger starts with this workspace or when you open this workspace in a session. This list includes user-defined projects and any auto-projects where you have saved the settings.

For more details, see the chapter describing workspaces in *RealView Developer Kit v2.2 Debugger User Guide*.

1.11   Projects in Connection Properties

You can configure a debug target, using the Connection Properties window, so that one or more projects open automatically as soon as you connect to that target. The projects can be user-defined projects or auto-projects.

For more details, see the chapter describing configuring custom targets in *RealView Developer Kit v2.2 Debugger User Guide*.

       ARM DUI 0291C

Chapter 2
Working with Projects

This chapter introduces the specific tasks and information you require to manage user-defined projects and to create your own projects in RealView® Debugger. It contains the following sections:

- *Using the Project menu* on page 2-2
- *Opening a user-defined project* on page 2-6
- *Generated makefiles* on page 2-8
- *Project properties* on page 2-9
- *Adding files to a user-defined project* on page 2-10
- *Upgrading the project toolchain* on page 2-12
- *Closing a user-defined project* on page 2-15.

2.1 Using the Project menu

The Code Window **Project** menu offers project-level commands that apply to a single project. This section describes how to use the **Project** menu with one or more projects and forms the basis for the rest of this chapter. This section contains:

- *Active project*
- *Menu options*.

2.1.1 Active project

RealView Debugger enables you to have several projects open at any time. If you select a command from the **Project** menu, RealView Debugger applies the chosen operation to the *active* project.

In a debugging session, the active project is usually the last project you open. If you are connected to a debug target, then the active project is the last project that RealView Debugger opens and successfully binds to the connection. The name of the active project appears in the default Code window title bar.

——— **Note** ———

When working with multiple projects or Container projects, you might have to change the active project so that you can access the project properties, and perform project-level operations, from the Code window. See *Changing the active project* on page 10-4 for details on how to do this.

———————————

2.1.2 Menu options

The **Project** menu (shown in Figure 2-1) contains project control commands to create projects, redefine project settings, and control access to projects during your debugging session. Some options in the **Project** menu are not available unless at least one project is open or a source file is visible in the File Editor pane.

New Project...
Open Project...
Project Control...
Close Project...
Recent Projects ▸

Project Properties...   Alt+F7
Build-Tool Properties...

Add This File to Project
Add Files to Project...
Update Dependencies
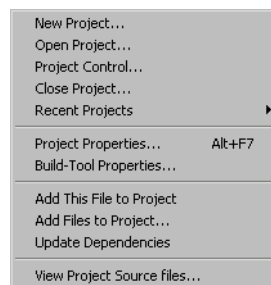
View Project Source files...

Figure 2-1 Project menu

The options available are:

New Project...

Displays the Create New Project dialog box where you can create a new user-defined project. See *Common steps for creating a user-defined project* on page 3-4 for details.

Open Project...

Displays the Select Project to Open dialog box where you can locate a user-defined project to open. A user-defined project is saved with the .prj extension.

You can also open a user-defined project by dragging the appropriate project settings file, with the .prj extension, from Windows Explorer and dropping it into the File Editor pane.

You can also open a project from the **Recent Projects** list.

Project Control...

Displays the Project Control dialog box where you specify actions to perform on one or more open projects. See *Using the Project Control dialog box* on page 6-2 for details.

Close Project...

Closes an open project. When two or more projects are open, a list selection box is displayed for you to choose which project or projects to close.

Recent Projects

Displays a list of up to nine user-defined projects that you have used in this or previous debugging sessions. On first opening RealView Debugger, this option is grayed out. Each project you create is added to this list, and the list is saved in your start-up file.

Project Properties...

Displays the Project Properties window where you can view and change project settings.

You can also display the Project Properties window by selecting **Debug** → **Set Source Search Path...**.

The project properties are saved in the project settings file (*.prj or *.axf.apr). The entries are populated when you first create the project, but you can amend them as required. If you load an image directly, you

can use this option to access project properties for the associated auto-project, created by RealView Debugger. See Chapter 4 *Using the Project Properties Window* for details.

Build-Tool Properties...

Displays the Build-Tool Properties window to enable you to specify the location of development tools accessible to RealView Debugger for building. This option is not used for auto-projects.

It is recommended that you define the build tools before creating your first project. See *Managing your build tools* on page 7-5 for details.

Add This File to Project

Adds the source file that is currently selected in the File Editor pane to the active user-defined project. This option is enabled when a source file is selected in the File Editor pane and a user-defined Standard or Library project is open.

For Custom projects, you must edit your own makefile to add files.

Add Files to Project...

Displays the Select file(s) to Add dialog box to enable you to select one or more source files to add to the active user-defined project. You can also select files from your personal favorites list. This option is enabled when one or more user-defined Standard or Library projects are open.

For Custom projects, you must edit your own makefile to add files to the project.

Update Dependencies
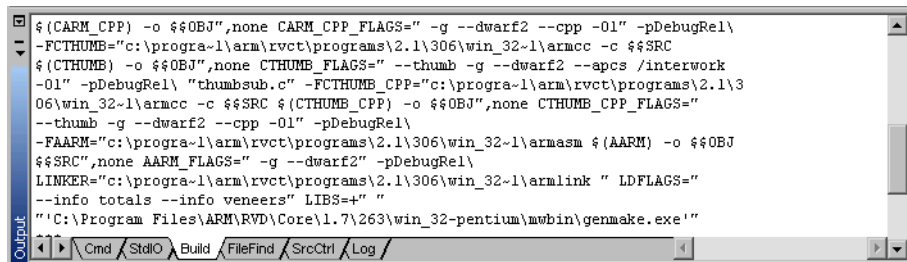
Updates dependencies for the active user-defined project. This option is enabled when a user-defined project is open that contains a makefile.

View Project Source files...

Displays the Project Source View selection box. The display list shows those source files defined in the makefile for the active user-defined project. Select a source file from the list and click **Edit** to open the file in the File Editor pane. This option is enabled when a user-defined project is open that contains a makefile.

You can also view source files for a project using the Process Control pane. See *Managing projects in the Process Control pane* on page 6-5 for details.

If you select certain options from the **Project** menu, for example **Add Files to Project...** or **Update Dependencies**, the project makefile is regenerated. When this happens, the Output pane switches to the **Build** tab and displays details about the makefile creation process, shown in Figure 2-2.

```
$(CARM_CPP) -o $$OBJ",none CARM_CPP_FLAGS=" -g --dwarf2 --cpp -O1" -pDebugRel\
-FCTHUMB="c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armcc -c $$SRC
$(CTHUMB) -o $$OBJ",none CTHUMB_FLAGS=" --thumb -g --dwarf2 --apcs /interwork
-O1" -pDebugRel\ "thumbsub.c" -FCTHUMB_CPP="c:\progra~1\arm\rvct\programs\2.1\3
06\win_32~1\armcc -c $$SRC $(CTHUMB_CPP) -o $$OBJ",none CTHUMB_CPP_FLAGS="
--thumb -g --dwarf2 --cpp -O1" -pDebugRel\
-FAARM="c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armasm $(AARM) -o $$OBJ
$$SRC",none AARM_FLAGS=" -g --dwarf2" -pDebugRel\
LINKER="c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armlink " LDFLAGS="
--info totals --info veneers" LIBS=+" "
"'C:\Program Files\ARM\RVD\Core\1.7\263\win_32-pentium\mwbin\genmake.exe'"
```

`◄ ► \ Cmd / StdIO \ Build / FileFind / SrcCtrl / Log /`

Figure 2-2 Rebuilding an application

Do not try to add any files to your project until this process is complete and the results are visible in the Output pane.

2.2   Opening a user-defined project

If you have started RealView Debugger, you can begin to use many features of the debugger, for example editing source code and building projects. However, to begin debugging images you must connect to a suitably configured debug target.

The examples in this section assume that you are using a Typical installation and that the software has been installed in the default location. If you have changed these defaults, or set the environment variable RVDEBUG_INSTALL, your installation differs from that described here.

——— **Note** ———

It is not necessary to connect to a debug target to follow these examples. However, they include instructions to make a connection, using RealView ICE Micro Edition (RVI-ME), to demonstrate how RealView Debugger binds projects. A similar procedure may be used for an RVI unit.

————————————

To open an existing user-defined project:

1.   Start RealView Debugger.

2.   Select **File → Connection → Connect to Target...** from the main menu to display the Connection Control window to make your first connection. For details on using this window, see the chapter describing how to connect to targets in *RealView Developer Kit v2.2 Debugger User Guide*.

3.   Connect to your target using RVI-ME.

4.   Select **Project → Open Project...** from the default Code window main menu. This displays the Select Project to Open dialog box.

5.   Locate the required project settings file from the \Examples directory, that is \dhrystone\dhrystone.prj, and click **Open** to load the project details into RealView Debugger.

      The project directory becomes the current working directory. This enables you to find new source files, or to set up build options more quickly.

2.2.1   Default binding

The dhrystone example project specifies ARM-C22 as the toolchain associated with the target application. Because you are connected to an ARM® target, RealView Debugger attempts to bind the project using default binding. If the project binds successfully, the default Code window title bar is updated with the project name to show that it is bound to the current connection, for example:

```
RVDEBUG(dhrystone) = @ARM946E-S_0:ARM-ARM-USB [Unattached]
```

Because RealView Debugger has the image details, the File Editor pane contains a hyperlink ready to load the associated image.

If you open a second project that specifies the same toolchain, for example `...\primes\primes.prj`, RealView Debugger gives you the option to unbind the first project and bind the second. In this example, click **Cancel**. This opens the `primes` example project but maintains `dhrystone` as the active project and the current binding remains unchanged.

See Chapter 9 *Project Binding* for more details on binding operations and how to control project binding.

2.2.2 Upgrading the project

RealView Debugger determines the location of your ARM build tools automatically (see *Managing your build tools* on page 7-5 for details). If you open a project and a later version of the toolchain exists, RealView Debugger prompts you to upgrade the project.

It is recommended that you upgrade the project. If you do not upgrade the project now, you can do so later if required. See *Upgrading the project toolchain* on page 2-12 for more details.

2.3   Generated makefiles

Project makefiles are generated when you do any of the following:

- create a new user-defined project and then save and close the Project Properties window

- edit and close the Build-Tool Properties window

- edit and close the Project Properties window

- edit and resave a user-defined project

- add source files to a user-defined project from the default Code window.

You can also force makefiles to be regenerated by deleting them from the project base directory and then rebuilding the image. Where appropriate, RealView Debugger generates a makefile for each build target configuration in your project (see Chapter 8 *Managing Build Target Configurations*).

RealView Debugger displays the details of the makefile generation process in the **Build** tab of the Output pane (see the example in Figure 2-2 on page 2-5).

2.3.1   Viewing the makefile

You can view the makefile that is generated using RealView Debugger in several ways:

- Select **File** → **Open...** from the default Code window main menu. This displays the Select File to Open dialog box, where you can locate the makefile and open it in the File Editor pane for viewing.

- Open Windows Explorer and navigate to the project base directory. Drag the makefile and drop it into the File Editor pane where it opens for viewing.

- Start a text editor of your choice and open the makefile for viewing.

Do not make any changes to the generated makefiles because they are overwritten when the files are next generated. It is recommended that you always use the Project Properties window to set up your preferences. However, you can also edit the template file that RealView Debugger uses to generate the makefile, for example `gen_arx.mk` used for building with ARM compilation tools.

2.4   Project properties

When you first create a project, RealView Debugger creates the project settings file and completes the entries to provide the project definition. RealView Debugger dynamically updates this file as you amend the project properties.

The name of the project settings file is defined by the project name and it is stored in the project base directory specified when you created the project. In general, the project settings file remains in this location. However, RealView Debugger uses relative pathnames so you can change the base directory for the project if required.

By default, the executable built by the project also uses the project name as the image name, for example dhrystone.axf. Although your project filename can be different from your image name, it might be advantageous to use the same name for your project settings file as your image name when developing your applications.

To view project properties for the active project dhrystone.prj, select **Project → Project Properties...** from the default Code window main menu. This displays the Project Properties window.

See Chapter 4 *Using the Project Properties Window* for details on entries in the Project Properties window.

2.5   Adding files to a user-defined project

You can add files to the active project from the:

- **Project** menu
- Project Properties window.

If the Project Properties window is visible, selecting an option from the **Project** menu might display an error dialog box instructing you to use the window to make the update. You must also wait for any build process to complete before trying to add files to the active project.

This section describes:

- *Adding files using the Project menu*
- *Adding files using the Project Properties window* on page 2-11.

2.5.1   Adding files using the Project menu

With a project open, you can add source files to the project and so update the project properties automatically:

1.   Choose the menu option according to the location of the file you want to add:

- Select **Project → Add This File to Project** to add the file currently displayed in the File Editor pane to the list of sources specified for the project.

- Select **Project → Add Files to Project...** to display the Select File(s) to Add dialog box where one or more files can be located and added to the project.

The relative pathname of the file is added to the Sources group in the top-level COMPILE or ASSEMBLE group. If this is the first update to the Sources group, an asterisk is appended to the beginning of the group name.

The Project Properties window is not displayed, but you can view the change by opening the window as described in *Project properties* on page 2-9.

If your project has more than one COMPILE or ASSEMBLE group enabled, a list selection box is displayed that lists the groups for the type of file being added:

- for files with extension .c, .cpp, .cc, or .cxx, the list shows enabled COMPILE groups

- for files with extension .s, .src, or .asm, the list shows enabled ASSEMBLE groups.

Figure 2-3 on page 2-11 shows example COMPILE groups for a dhrystone project.
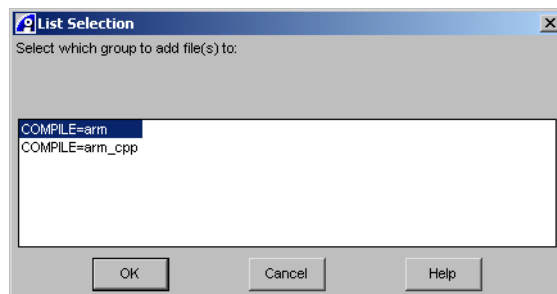

Figure 2-3 Add project source file selection box

2.   Choose the required group, for example COMPILE=arm and, then click **OK** to close the dialog box.

The **Build** tab shows the makefile being regenerated.

3.   When the makefile has been regenerated, select **Tools** → **Build...**, to rebuild the image.

———— **Note** ————

If you select files of different types, for example, ∗.c and ∗.s files, then RealView Debugger adds the files to the appropriate group, even if that group is disabled. In this example, the ∗.c files are added to the COMPILE=arm group, and the ∗.s files are added to the ASSEMBLE=arm group.

2.5.2   Adding files using the Project Properties window

For examples of adding files using the Project Properties window, see Chapter 5 *Customizing Projects*:

•   *Adding source files to a project* on page 5-6

•   *Adding object files* on page 5-15

•   *Adding library files* on page 5-17.

Also, see *Specifying paths to include files* on page 5-19 if you have project-specific include files in a different directory from your main source files.

2.6 Upgrading the project toolchain

If you have upgraded your build tools, you can upgrade your projects to use the new toolchain. For example, if you currently have *RealView Developer Kit* (RVDK) and you upgrade to a later version, you can upgrade your existing RVDK projects to use the new toolchain. You can also upgrade ADS projects to RVDK.

RealView Debugger determines the location of your ARM build tools automatically (see *Managing your build tools* on page 7-5 for details). If you open a project and a later version of the toolchain exists, RealView Debugger prompts you to upgrade the project. The way in which this happens depends on whether or not the toolchain with which the project was built is installed on your system.

When you open a project, RealView Debugger examines it to find out which toolchain was used to create it. At this stage, the project becomes the active project, but remains in the validation stage instead of being fully opened. RealView Debugger then performs one of the following actions:

• If the original toolchain is no longer present on your system and an upgrade is available for the project, you must upgrade the project before you can use it. RealView Debugger displays the Upgrade Project selection box, shown in Figure 2-4.
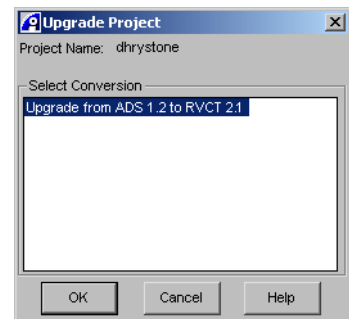
Figure 2-4 Upgrade Project selection box

The name of the project you are about to upgrade appears at the top of the selection box and the display list shows available upgrades. Do one of the following:

— If you want to upgrade the project, select the required conversion and click **OK**. The project is upgraded and RealView Debugger completes the validation process. The project is then open and ready for use. A backup copy of the old project settings file is created automatically in the same location.

— If you do not want to upgrade the project, click **Cancel**. The validation process is not completed and the project is closed.

• If the original toolchain is present and an upgrade is available for the project, you can use the project whether or not you upgrade it. RealView Debugger therefore offers you the appropriate binding options before prompting you to upgrade the project (see Chapter 9 *Project Binding* for information on binding).

RealView Debugger then displays the Upgrade Project ToolChain dialog box, shown in Figure 2-5.
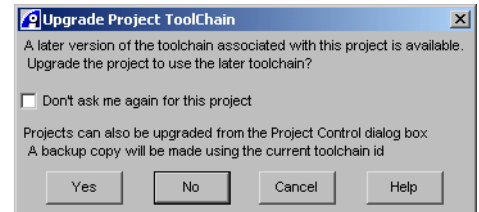

Figure 2-5 Upgrade Project ToolChain dialog box

Do one of the following:

— Click **Yes** if you want to upgrade your project. This displays the Upgrade Project selection box, shown in Figure 2-4 on page 2-12. A backup copy of the old project settings file is created automatically in the same location.

— Click **No** if you do not want to upgrade your project now. In this case, RealView Debugger remembers the state of the **Don't ask me again for this project** check box. If you want to upgrade the project later, then leave this check box unchecked.

RealView Debugger completes the validation process and the project is open and ready for use.

——— **Note** ———

If you select **Don't ask me again for this project**, and you want to upgrade that project later, you can use the Project Control dialog box to upgrade the project manually. See on page 6-2*Using the Project Control dialog box* on page 6-2 for details.

————————————

— Click **Cancel** if you do not want to upgrade your project. The **Don't ask me again for this project** checkbox is ignored, and the next time you open the project, you are asked if you want to upgrade it.

RealView Debugger completes the validation process and the project is open and ready for use.

• If the original toolchain is not present and an upgrade is not available for the project, you cannot use the project and an error message is displayed.

When a project is successfully upgraded to the latest toolchain, the dialog box is not displayed when you next open that project, unless you upgrade your build tools again. If you upgrade your project to an intermediate toolchain, the dialog box is displayed the next time you open the project. Click **No** and check **Don't ask me again** if you do not want to upgrade to the latest toolchain.

———— **Note** ————

If you upgrade a project for which you do not have the tools installed, you must force the binding behavior. To do this, either close the project and reopen it, or use the Project Control dialog box.

———— **Note** ————

It is recommended that you rebuild your project after upgrading it.

2.7 Closing a user-defined project

As you change a user-defined project, add new files, or update dependencies, the project properties are updated and saved automatically. It is not necessary to close your project to update the associated project settings file.

Select **Project** → **Close Project...** from the default Code window main menu to close an open project. If the associated Project Properties window is displayed, this closes automatically when the project closes. You are warned about any unsaved changes to the Project Properties window before it closes.

If you have more than one project open, RealView Debugger displays a selection box where you can specify which project to close, for example dhrystone. If you have two projects open, the second project, for example primes, becomes the active project when the first project closes.

When you close a project, the default Code window title bar is updated to show the new active project, where applicable. The current working directory remains as defined by the last file access. You can change this by resetting it or when you open another project from a different location.

2.7.1 Project binding

When you close a bound project, it unbinds automatically. If you had two projects open, the second project, for example primes, does not bind by default because this only happens when a project opens to a connection. You can, however, rebind the project if required.

———— **Note** ————

If you are connected when you close the project, any close commands you have specified are executed. If you are not connected when the project closes, these commands are not run.

See Chapter 9 *Project Binding* for more details on binding operations and how to rebind a project.

2.7.2 Working with images

If you close a user-defined project and the image is not loaded, RealView Debugger removes all image details. This clears the hyperlink in the File Editor pane.

If you close a user-defined project where the image is loaded, RealView Debugger prompts you to unload the image. Click **Yes** to unload the image and remove all image details.

If you do not unload the image, RealView Debugger searches for a saved auto-project to provide project properties for the loaded image. If no file exists for the image, RealView Debugger creates an in-memory auto-project to use in this session. This binds to the connection by default.

 ARM DUI 0291C

Chapter 3
Creating New Projects

This chapter introduces the specific tasks and information you require to manage user-defined projects and to create your own projects in RealView® Debugger. It contains the following sections:

- *Creating a new user-defined project* on page 3-2
- *Common steps for creating a user-defined project* on page 3-4
- *Steps for creating a Standard or Library project* on page 3-6
- *Steps for creating a Custom project* on page 3-8
- *Steps for creating a Container project* on page 3-12
- *Steps for copying an existing user-defined project* on page 3-14
- *Customizing and building your project* on page 3-15
- *Merging auto-project settings into a project* on page 3-16.

3.1 Creating a new user-defined project

There are two types of project in RealView Debugger:

User-defined projects

>Projects that you create and set up.

Auto-projects

>Projects that RealView Debugger creates or uses automatically when you load an image directly.

The procedure for creating a user-defined project consists of three stages:

1. Use the Create New Project dialog box to set up the new project and define the project type. This stage is common to all types of user-defined project.

2. Use the project-specific dialog box to configure the settings for your new project.

3. Customize and build the project.

If you have modified and saved settings for an auto-project, then you can merge these settings into a new user-defined project during the second stage of the creation procedure.

———— **Note** ————

If you do want to merge auto-project settings, ensure that you read the explanation in *Merging auto-project settings into a project* on page 3-16 before you start to create the new project.

This rest of this chapter describes how to create a new project. It explains how to complete the common steps (the first stage), how to create each type of project (the second stage), and how to complete the build process (the third stage). The chapter ends by describing how to merge auto-project settings:

- *Before you start* on page 3-3
- *Common steps for creating a user-defined project* on page 3-4
- *Steps for creating a Standard or Library project* on page 3-6
- *Steps for creating a Custom project* on page 3-8
- *Steps for creating a Container project* on page 3-12
- *Steps for copying an existing user-defined project* on page 3-14
- *Customizing and building your project* on page 3-15
- *Merging auto-project settings into a project* on page 3-16.

3.1.1 Before you start

Before you follow the procedures described in the rest of this chapter:

1.   Start RealView Debugger.

2.   Connect to your target using RVI or RVI-ME.

3.   Ensure that you have no projects open. You can create new projects with open projects but this affects binding behavior.

4.   Set up your build tools as described in *Managing your build tools* on page 7-5.

Depending on your installation, RealView Debugger locates toolchains for projects automatically, for example *RealView Compilation Tools* (RVCT). These are used when you build any new user-defined project that you create, or if you rebuild an example project. However, you can override these settings in your project so that different tools are used for a project or for a particular set of files forming part of a project.

It is recommended that you define the location of your build tools before creating your first project. However, this is not essential. If you do not do this, RealView Debugger asks you to specify the toolchain to use when saving your first project. This is then used for all new projects that you create.

3.2   Common steps for creating a user-defined project

To create a new user-defined project:

1.   Select **Project** → **New Project...** from the default Code window main menu to display the Create New Project dialog box.

2.   Complete the entries in the Create New Project dialog box:

    Project Name

        Enter a name for your project in this data field.

        Check that any name entered here is not already in use for a project. Where a project with the same name already exists, RealView Debugger gives you the option to replace it.

    Project Base

        Specify the project base directory to use as the location of the project files.

        This data field might be preloaded with your RealView Debugger installation directory name as defined by your environment variable. This can be overridden.

        Click the folder icon to view the directory chooser, and select **<Select Dir...>** to specify a directory that is not in the recently used directory list.

        If the specified directory does not exist, RealView Debugger gives you the option to create it.

        If you are creating a Custom project, the project base directory must contain your makefile.

    Select Type of Project

        Select the type of project you want to create. For more details about the project types, see *Types of project* on page 1-3.

3.   Click **OK**.

    ─── **Caution** ───

    If you confirm that an existing project is to be overwritten, there is no undo and the contents of the first project settings file are lost.

    ────────

4.   RealView Debugger displays a dialog box to create the type of project you selected.

    Follow the steps described in the appropriate section to complete the entries:

    •   *Steps for creating a Standard or Library project* on page 3-6
    •   *Steps for creating a Custom project* on page 3-8

- • *Steps for creating a Container project* on page 3-12
- • *Steps for copying an existing user-defined project* on page 3-14.

5. Click **OK** to confirm the project details and close the dialog box for your project type.

6. Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-15.

3.3   Steps for creating a Standard or Library project

The steps described in this section assume that you have performed the first three steps described in *Common steps for creating a user-defined project* on page 3-4.

To create a Standard or Library project:

1.   Complete the entries on the Create Standard Project or the Create Library Project dialog box:

Project Name

This shows the name specified for the new project.

Toolchain

Choose the toolchain to use for this project from the drop-down list. RealView Debugger uses this to bind the project to all available debug targets that have the same processor type.

Sources (C/C++/Assembly) to build from

Click the directory icon to locate the sources used to build the executable image or library file. The source files are added to the source list box, and they are all selected.

Alternatively, enter a filename in the text box, then click the **Add** button. The file is added to the source file list, and is selected.

You can perform other operations on the source file list:

- Click **Del** to delete all selected files from the source file list.

- Click **Rep** to replace the selected file in the source file list with the file specified in the text box.

- Click **AllOn**, or **AllOff**, to select, or deselect, all files in the source file list.

————— **Note** —————

You do not have to add your source files here. You can create the project, and then add the source files later. See *Adding files to a user-defined project* on page 2-10 for details.

Support for C++ may be obtained as an upgrade on certain versions of RVDK, by means of an appropriate FLEX*lm* license.

Executable

RealView Debugger completes this field based on the project name, for example std_proj_1.axf or lib_proj_1.lib. Change this entry if required.

Alternatively, click on the directory icon to locate the required executable or library file.

Description

                Enter a text description for the new project, saved in the PROJECT group.

2.   Click **OK** to confirm the project details and close the Create Standard Project or the Create Library Project dialog box.

3.   Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-15.

3.4   Steps for creating a Custom project

The steps described in this section assume that you have performed the first three steps described in *Common steps for creating a user-defined project* on page 3-4.

This section also describes:
- *The MAKEFILE group* on page 3-10
- *Using your own make command* on page 3-10.

To create a Custom project:

1.   Complete the entries on the Create Custom Project dialog box:

Project Name

This shows the name specified for the new project.

Toolchain

Select the toolchain to use for this project from the drop-down list.

RealView Debugger uses this to bind the project to all available debug targets that have the same processor type.

Select type of Custom Project

Specifies how the image is built for the project and determines what additional information you must provide.

Make a makefile (your own makefile)

Uses the default `make` command together with your makefile to build the image.

For a Custom project, the project base directory must contain your makefile.

Run Command (your own tool/builder)

Uses your own build tool command to build the image. You must enter the command in the Command field.

If you want to use the command expansion controls, `$a`, `$e`, and `$f`, then you must specify the information in the corresponding field.

———— **Note** ————

To build your Custom project successfully, you must remove the `$e` control character, and use your own arguments as required. See *Using your own make command* on page 3-10.

————————————

No-Build (echo Arguments as message only)

If you do not want to build an image for this project.

                                       ARM DUI 0291C

Command

This data field contains the default make command used for the new project, for example, make -f $f $a $e.

You can also use the project path expansion control $p in the command, for example, make -f $p\$f. The path expansion control uses the path you specified in the Project Base data field, see *Common steps for creating a user-defined project* on page 3-4 for details.

See *Using your own make command* on page 3-10 for details on how to amend this default makefile command.

Use the **File Arg**, **Arguments**, and **Executable** fields to populate the makefile command line or set up the command manually.

———— **Note** ————

The Command field must be filled in even when the project specifies a no-build model, for example use a dummy entry such as dummy.

Do not enter any other command here, for example to run a batch file.

File Arg

This data field contains the name of the makefile, that is $f.

By default, the name of the makefile is the same as the name of the new Custom project. If your makefile has a different name, change the name in this field so that it is correct.

Arguments

This data field contains the arguments to the command, that is $a.

Executable

This data field contains the executable file to build, that is $e.

Description

This data field contains a text description for the new project, saved in the PROJECT group.

2. Click **OK** to confirm the project details and close the Create Custom Project dialog box.

3. Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-15.

3.4.1 The MAKEFILE group

When you create a Custom project, the project settings file contains a special group of settings, the MAKEFILE group, that can be used to override the default makefile and enable you to use your own make command. Table 3-1 describes the settings available in the MAKEFILE group.

Table 3-1 MAKEFILE group

Name	Description
Makefile	Specifies the makefile built for the project, if this project contains a makefile.
Application	Defines the target application program built for the project.
Arguments	Specifies the arguments to pass to the make or other build command.
Cwd	Specifies the working directory when running your build.
Command	Specifies your own make command that can be make, another build tool, or none if this is a no-build project. See *Using your own make command* for more details.

——— **Note** ———

For an auto-project, these settings are set up for a no-build project. Do not change the settings in this group for an auto-project.

You can create a user-defined project using the image for an auto-project, and choose to merge the auto-project settings into the user-defined project settings. The MAKEFILE group is not merged. For more details, see *Merging auto-project settings into a project* on page 3-16.

3.4.2 Using your own make command

In a simple project, the default build rule is to run the make utility. For a Custom project, you can run make or another build tool.

By default, the project toolchain defines the location of the make command if you do not specify your own command.

Your make command can include controls that are expanded by RealView Debugger. You can use this setting to create any command line you require.

Specify each control character as $ followed by a letter as in:

```
make -f $f $a $e $t $p
```

where:

$f   Is the name of the makefile from the `Makefile` setting (see Table 3-1 on page 3-10).

$a   Is the arguments list from the `Arguments` setting (see Table 3-1 on page 3-10).

$e   Depends on the action you select on the **Tools** menu:

       all   If you select **Tools → Build...**

       rebuild If you select **Tools → Rebuild All (Clean+Build)**

       clean   If you select **Tools → Clean (remove objects)**

       object_file

           If you select **Tools → Build This File**. This might include a path name that is relative to the project directory.

$t   Is the target filename that is built, or cleaned and built from the `Application` setting (see Table 3-1 on page 3-10).

$p   Is the project directory. This is the directory you specified when you created the project.

If the `Command` setting is empty then the command defaults to `make -f $f $a $e`.

——— **Note** ———

To successfully build your Custom project, remove the $e control character, and use your own arguments as required.

————————————

3.5   Steps for creating a Container project

The steps described in this section assume that you have performed the first three steps described in *Common steps for creating a user-defined project* on page 3-4.

To create a Container project:

1.   Complete the entries on the Create Container Project dialog box:

Project Name

This shows the name specified for the new project.

Sub-Projects (order defines build order)

Click the directory icon to locate the project files to use for the Container project. The project settings files are added to the project list box, and they are all selected.

Alternatively, enter a project filename in the text box, then click the **Add** button. The file is added to the project file list, and is selected.

You can perform other operations on the project file list:

- Click **Del** to delete all selected files from the project file list.
- Click **Rep** to replace the selected file in the project file list with the file specified in the text box.
- Click **AllOn**, or **AllOff**, to select, or deselect, all files in the project file list.

———— **Note** ————

You do not have to add your subprojects here. You can create the project, and then add the subprojects later using the Project Properties window. See *Working with Container projects* on page 10-7 for more details.

———————————————

Description

Enter a description for the project, saved in the PROJECT group.

2.   Click **OK** to confirm the project details and close the Create Container Project dialog box.

3.   Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-15.

The nature of Container projects means that if you have a Container project open, you are working on multiple projects at the same time. You can perform additional operations on the individual projects in a Container project. See *Working with Container projects* on page 10-7 for more details.

———— **Note** ————

Container projects can be nested but not recursive, that is a Container project can include other Container projects but must not include itself. However, the nested Container project has no makefile and so any build fails.

————————————

3.6   Steps for copying an existing user-defined project

The steps described in this section assume that you have performed the first three steps described in *Common steps for creating a user-defined project* on page 3-4.

To create a copy of an existing user-defined project:

1.   Locate the project settings file to copy on the Select Project to Copy from dialog box.

        By default, RealView Debugger begins the search in the destination location because this is now the current working directory.

2.   Click **Open** to copy the project details to the new project.

3.   If the project you are using to create the new project has associated subdirectories, you are prompted to copy the directory tree structure. In general, it is recommended that you copy the existing tree structure for the new project:

        •   Click **Yes** to copy the directory tree to the new project. RealView Debugger replicates the entire contents starting at the location of the specified `.prj` file.

        •   Click **No** if you do not want to copy the tree.

4.   If you are copying a Container project, RealView Debugger displays a list of any subprojects, specified in the project, that could not be opened. Make a note of these projects. You can update these projects later, or delete them, using the Project Properties window. See *Working with Container projects* on page 10-7 for more details.

        Click **OK** to confirm the project details and close the Select Project to Copy from dialog box.

5.   Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-15.

                                                                 ARM DUI 0291C

3.7 Customizing and building your project

When you close the dialog box for your project type, RealView Debugger:

1. Creates the `.prj` file in the specified location.

2. Opens the project into the debugger.

3. Binds the project to the current connection.

    If a project is already bound to the connection, RealView Debugger gives you the option to unbind the first project and bind the second. Although it is not necessary, bind the new project to complete the creation procedure.

4. Updates the default Code window title bar to show the active project.

5. Displays the Project Properties window for you to customize your project.

    The entries shown in this window depend on the type of project you create.

To complete your new project:

1. Make any changes you require in the Project Properties window, see Chapter 4 *Using the Project Properties Window* for details.

2. Select **File → Save and Close** to close the Project Properties window and generate the makefile(s) for the new project.

3. Select **Tools → Build...** from the default Code window main menu to build the active project.

3.8   Merging auto-project settings into a project

If you have an image and you have modified, and saved, an auto-project file associated with that image, you can merge those settings with the settings for a new project.

————— **Note** —————

Merging only gives you the option to incorporate the SETTINGS group from your auto-project. If you change settings in any other group, for example the Command_Open_Close commands in the PROJECT group, these are lost when the auto-project merges.

—————————————————

To merge an auto-project you must have:

*   an image, for example \test_examples\test_image.axf

*   a saved auto-project for the image, for example
    \test_examples\test_image.axf.apr.

By definition, an auto-project is a no-build project and, therefore, uses no build model. You can merge an auto-project into the following types of project:

Standard   Create a new Standard project to merge the settings and use a build model.

Custom   Create a new Custom project to merge the settings and use your own makefile or build tools. This project uses no build model.

See *Using a build model* on page 7-14 for more details on build models.

This section describes:
*   *Merging options*
*   *Common steps for merging auto-project settings* on page 3-17
*   *Steps for merging settings into a Standard project* on page 3-18
*   *Steps for merging settings into a Custom project* on page 3-18
*   *Customizing and building your project* on page 3-19.

3.8.1   Merging options

If you have made changes to load-related values in the SETTINGS group of your auto-project, you can choose whether to merge these settings into the properties of the new project. During the creation process, a list selection box is displayed that enables you to specify the merge, shown in Figure 3-1 on page 3-17.
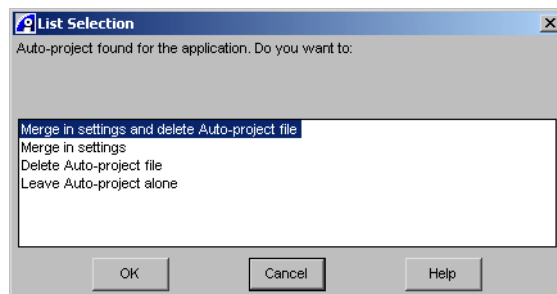
       ARM DUI 0291C

Figure 3-1 Merging options selection box

The merging options are:

Merge in settings and delete Auto-project file

        Merges the SETTINGS group from the auto-project into the new project and then deletes the auto-project.

Merge in settings

        Merges the SETTINGS group from the auto-project into the new project but keeps the auto-project.

Delete Auto-project file

        Deletes the auto-project without merging the SETTINGS group.

Leave Auto-project alone

        Keeps the auto-project without merging the SETTINGS group.

        —— **Note** ——

        Choose this if you have not saved any settings for your auto-project. This is the same as clicking **Cancel**.

3.8.2 Common steps for merging auto-project settings

Create the user-defined project you require:

1.   Follow the first three steps in *Common steps for creating a user-defined project* on page 3-4 with these entries:

    a.   Enter the name of your project in the Project Name data field.

    b.   Enter the directory containing your auto-project file in the Project Base data field, for example \test_examples.

> c. Select the type of project to create, either:
>   - **Standard**
>   - **Custom.**

2. RealView Debugger displays a dialog box to create the type of project you selected.

   Follow the steps described in the appropriate section to complete the entries:
   - *Steps for merging settings into a Standard project*
   - *Steps for merging settings into a Custom project*.

3. Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-19.

3.8.3 Steps for merging settings into a Standard project

To merge your auto-project settings:

1. Complete the entries in the Create Standard Project dialog box as described in the first step in *Steps for creating a Standard or Library project* on page 3-6.

   You must specify the full pathname of the image associated with the auto-project in the Executable data field, for example \test_examples\test_image.axf.

2. Click **OK** to display the Project Properties window.

3. If RealView Debugger locates the auto-project for the specified image, it displays a list selection box where you can choose the merging option (see *Merging options* on page 3-16 for details):

   a. Select the required option from the list.

   b. Click **OK** to update the new project settings file as requested. If selected, the auto-project file is also deleted.

4. Make any changes to the project properties to customize the new project. See *Customizing and building your project* on page 3-19.

3.8.4 Steps for merging settings into a Custom project

To merge your auto-project settings:

1. Complete the entries in the Create Custom Project dialog box as described in the first step in *Steps for creating a Custom project* on page 3-8.

   You must specify the full pathname of the image associated with the auto-project in the Executable data field, for example \test_examples\test_image.axf.

2. Click **OK**.

3. If RealView Debugger locates the auto-project for the specified image, it displays a list selection box where you can choose the merging option (see *Merging options* on page 3-16 for details):

    a. Select the required option from the list.

    b. Click **OK** to update the new project settings file as requested. If selected, the auto-project file is also deleted.

4. Make any changes to the project properties to customize the new project. See *Customizing and building your project*.

3.8.5 Customizing and building your project

When you close the merging options selection box, RealView Debugger:

1. Creates the .prj file in the specified location.

2. Opens the project into the debugger.

3. Binds the project to the current connection, if possible.

4. Updates the default Code window title bar to show the active project.

5. Displays the Project Properties window for you to customize your Standard project. If you merge into a Custom project, select **Project → Project Properties...** to display this window.

To complete your new project:

1. Make any changes you require in the Project Properties window, see Chapter 4 *Using the Project Properties Window* for details.

2. Select **File → Save and Close** to close the Project Properties window and generate the makefile(s) for the new project, where applicable.

3. Select **Tools → Build...** from the default Code window main menu to build the active project.

    ———— **Note** ————

    When you build a new image, the image is placed in the directory defined by the active build model for the project, for example \test_examples\Debug. Your original image remains in the project base directory, for example \test_examples.

ARM DUI 0291C

Chapter 4
Using the Project Properties Window

This chapter describes the contents of the Project Properties window when you are working with projects in RealView® Debugger. It contains the following sections:

4.1 Viewing project properties

To examine project settings for the active project, select **Project → Project Properties...** from the default Code window main menu. This displays the Project Properties window shown in Figure 4-1.
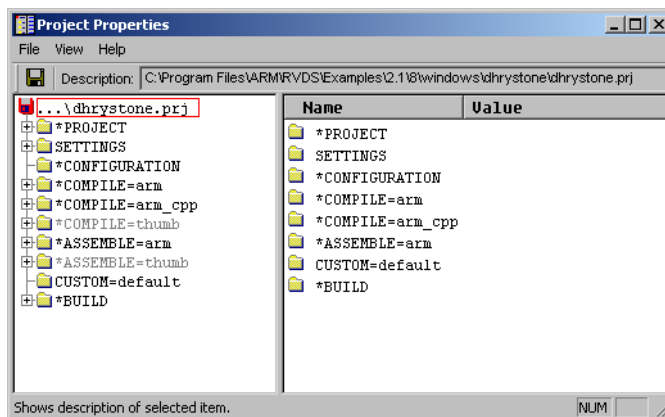

Figure 4-1 Project Properties window

You can also display the Project Properties window:

- by selecting **Debug → Set Source Search Path...**

- from the Process Control pane if you are connected to a debug target. See *Managing projects in the Process Control pane* on page 6-5 for more details.

Use the Project Properties window to view and change the project settings file. This contains the settings values, or rules, that describe a project. In some cases, these values are preset and cannot be changed. In other cases, you can amend the preloaded settings, or define your own values to customize your project.

If an entry has been changed from the default setting, an asterisk (*) is appended to the group name to show that the contents have been updated. You can choose to view only those settings that you have changed from the default. To do this, select **View → Show Default Entries** to disable the menu option. When the menu option is enabled, all settings are available for viewing.

――――― **Note** ―――――

The settings shown in the Project Properties window vary depending on the type of project. For example, if your project is a Custom project using a custom makefile, the window contains the PROJECT, SETTINGS, and MAKEFILE groups.

4.2 Project Properties window

The Project Properties window enables you to customize your project in the same way that you configure other settings files, such as build tool properties and workspace settings.

Select **Project** → **Project Properties...** from the default Code window main menu to display the Project Properties window, shown in Figure 4-2.
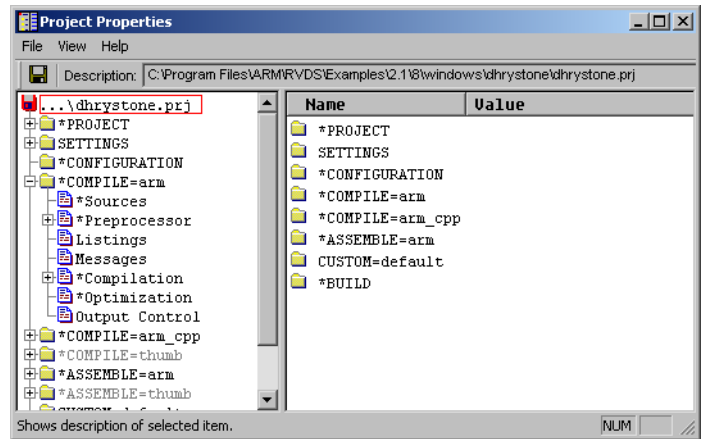

Figure 4-2 Groups in the Project Properties window

Figure 4-2 shows the properties for a Standard project, for example dhrystone.prj. Other types of project have a different set of properties.

The main interface components of this window are:

Main menu This contains:

      File Displays the **File** menu where you can save the project settings file after you have made changes.

      View Displays the **View** menu to toggle the display to show all the settings or only those that have been edited.

      Help Displays the online **Help** menu.

Save icon Click this icon to save the project settings file to disk.

      The name of the current file is shown as the first entry in the left pane.

Description This field displays a one-line description about an entry selected in the List of Entries pane or the Settings Values pane.

This section describes this window in more detail:

- *List of Entries and Settings Values*
- *Project settings groups* on page 4-5
- *Working with project settings groups* on page 4-5.

4.2.1 List of Entries and Settings Values

The left pane of the Project Properties window, the List of Entries pane, shows project settings as a hierarchical tree with node controls. The right pane, the Settings Values pane, displays the contents of any group that you select in the left pane. Groups of settings are associated with an icon to explain their function:

Red disk   This is a container disk file.

RealView Debugger uses this, for example, to specify an include file.

Yellow folder

This is a parent group containing other groups (*rules pages*) and/or entries.

Rules page   A rules page is a container for settings values that you can change in the right pane. When unselected, the pencil disappears (see Figure 4-2 on page 4-3).

This icon only appears in the left pane.

The first setting in the List of Entries pane, the left pane, is the name of the project settings file, for example `...\dhrystone.prj`. When selected, the Description field shows the full pathname of this file.

If you click on an entry in the left pane, a red box is drawn around it and the Description field is updated. At the same time, the right pane, the Settings Values pane, is updated to show the contents of the highlighted group (shown in Figure 4-2 on page 4-3).

4.2.2 Project settings groups

Table 4-1 shows the groups that are available for the different project types.

Table 4-1 Project settings groups

Project type	Groups	
Standard	PROJECT SETTINGS CONFIGURATION COMPILE	ASSEMBLE CUSTOM BUILD
Library	PROJECT SETTINGS CONFIGURATION COMPILE	ASSEMBLE CUSTOM BUILD_LIB
Custom and auto-project	PROJECT SETTINGS MAKEFILE	
Container	PROJECT	

4.2.3 Working with project settings groups

Each project group defines that part of the build model covered by the contents. For example, in a Standard project the CONFIGURATION group defines the build target configurations and the COMPILE group defines the compilation stage.

Where appropriate, a project might contain multiple groups, for example an interworking project that uses both ARM® and Thumb® code. Use unique names to identify multiple groups for this type of project.

If a group is disabled, the entry in the left pane is grayed out, shown in Figure 4-2 on page 4-3. You can delete disabled groups but this is not necessary. Any group that is disabled is ignored for the project, even if entries in the group are set (see *Disabling a group* on page 4-6 for details). If you delete a disabled group, it might be harder to change the project later.

Where permitted, groups can be deleted, renamed, or copied. Making a copy of an existing group creates a group that can then be edited in the usual way. If the group you are copying is marked by an asterisk, the copy is also marked in the same way.

You can also define a new group so that you can configure the build model to your specification and choose appropriate names for the groups in the project. Making a new group creates a new container for the default settings. See Chapter 5 *Customizing Projects* for examples of how to work with groups.

Disabling a group

To disable a group in the Project Properties window:

1.   Right-click on the group to be disabled, for example `*COMPILE=arm`, and select **Explore** from the context menu.

2.   Right-click on the `Disable` setting in the Settings Values pane and select **True** from the options list.

3.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

4.   Select **Tools → Build...** to rebuild the application.

If you disable a group, the entry in the left pane is grayed out but it does not appear as an entry in the right pane of the Project Properties window, shown in Figure 4-2 on page 4-3. You can view and change entries in a disabled group in the usual way, see Chapter 5 *Customizing Projects* for details.

To enable the group again:

1.   Right-click on the disabled group, for example `*COMPILE=arm`, and select **Explore** from the context menu.

2.   Right-click on the `Disable` setting in the Settings Values pane and select **False** from the options list.

3.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

4.   Select **Tools → Build...** to rebuild the application.

———— **Note** ————

See the RealView Debugger online help topic *Changing Settings* for details on all the entries in the Project Properties window.

————————————————

4.3   Viewing the Configuration Summary

Use the Configuration Summary to see a display of the switches that RealView Debugger is passing to the compilation tools (compiler, assembler, and linker) for each build target configuration in a project. This information is displayed in a read-only, tabbed window. When you make a change that affects the switches of a tool, the list of switches in the Configuration Summary window for that tool is updated immediately.

To view the list of switches:

1.   In the Project Properties window, select the group that relates to the tool of interest, that is:

•   a COMPILE group to view the switches for the compiler

•   an ASSEMBLE group to view the switches for the assembler

•   the BUILD group to view the switches for the linker of a Standard project

•   the BUILD_LIB group to view the switches for the linker of a Library project.

2.   Select **View** → **Configuration Summary** to see the Configuration Summary window. This is placed below the Project Properties window by default. Figure 4-3 shows an example configuration summary for the ARM C compiler.
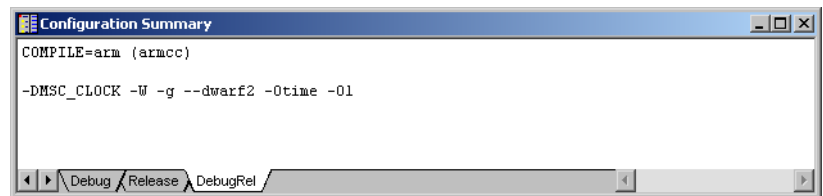
Figure 4-3 Example Configuration Summary window

3.   To view the switches for a specific build target configuration, select the tab for that configuration.

4.   To close the Configuration Summary window, click the **X** button, shown in the top-right corner of the window in Figure 4-3.

———— **Note** ————

If you close the Project Properties window when the Configuration Summary window is open, then both windows close. When you next open the Project Properties window, the Configuration Summary window is also displayed.

Chapter 5
Customizing Projects

You can change your project configuration by editing the project settings file using the RealView® Debugger Project Properties window. This chapter describes examples of how to customize your projects. It contains the following sections:

5.1 Working with the examples

This chapter contains examples of making changes to the user-defined project dhrystone.prj installed in the *install_directory*\RVDK\Examples\... directory. You might want to make a backup of the project base directory before following the examples so that the default files and settings can be restored.

——— **Caution** ———

The changes described here conflict with the default build target configurations as specified in the example project. The examples in this chapter are included only to show how to amend project and build-tool properties.

It is recommended that you read this section before following the examples:
- *Before you start*
- *Connecting to a debug target* on page 5-3
- *Using genmake* on page 5-3
- *Changing settings* on page 5-4.

5.1.1 Before you start

To start working with the example dhrystone project:

1. Make a copy of the project base directory if necessary.

2. Start RealView Debugger.

3. Select **Project → Open Project...** from the default Code window main menu.

4. Locate the project settings file dhrystone.prj and click **Open**.

You can use any project to complete this chapter. However, RealView Debugger determines the location of your ARM® build tools automatically. If you open a project and a later version of the toolchain exists, RealView Debugger prompts you to upgrade the project. You must upgrade the project before completing these examples. See *Upgrading the project toolchain* on page 2-12 for more details.

——— **Note** ———

If you are using the example dhrystone project, you can still follow the steps for adding sources, but you must then remove the duplicate source files before continuing with the other tasks. See *Removing source files from a project* on page 5-7.

5.1.2 Connecting to a debug target

These examples assume that you are not connected to a debug target. This means that RealView Debugger does not try to set default binding (see Chapter 9 *Project Binding* for more details). However, if you complete the source search paths example (*Specifying source search paths* on page 5-9), or the breakpoints example (*Specifying breakpoints* on page 5-23), or the interworking example (*Interworking ARM and Thumb* on page 5-29), you must connect and load an image to test your changes to the project. Ensure that you disconnect before trying any of the other examples in this chapter.

5.1.3 Using genmake

When you configure and save your project settings file, RealView Debugger uses a template file named gen_***.mk to generate the required makefile for your project (see *Makefiles* on page 1-12 for details).

These examples use the template \etc\gen_arx.mk for building executable files and libraries with ARM compilation tools to run on the ARM family of processors. This template provides the RVDK and ADS specific makefile layouts for the genmake utility used by RealView Debugger.

When using the genmake utility, be aware that some Windows-specific code causes genmake to translate separators. This can be avoided by using $ as an escape character to prevent the translation of / to \, shown in the example in Figure 5-1.
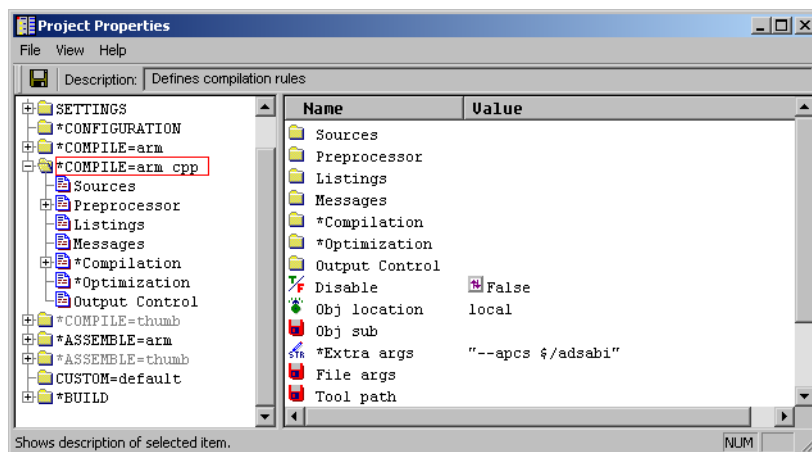

Figure 5-1 Using an escape character

5.1.4   Changing settings

These examples describe how to change project settings, using the Project Properties window. For details on the layout of this window see *Project Properties window* on page 4-3. These examples assume you are familiar with the procedures described in the online help topic *Changing Settings*.

       ARM DUI 0291C

5.2   Changing build tools for a specific user-defined project

When you install RealView Debugger for the first time, it determines the location of your build tools to use for all user-defined projects that you create. However, you can override this setting or specify a different tool for selected projects.

To change the compiler for a project:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the required group, in the List of Entries pane, the left pane, that defines the build stage that you want to modify, for example *COMPILE=arm, and select **Explore** from the context menu.

      You can also expand, and collapse, groups in the List of Entries pane by clicking on the plus sign, or the minus sign, at each node in the tree. If you double-click on the group name, this expands the group and displays the contents in the Settings Values pane.

3.   Right-click on the Tool_path setting in the Settings Values pane, the right pane, and select **Edit as Filename** from the context menu.

4.   Use the Enter New Filename dialog box to locate the compiler for this project.

5.   Click **Save** to confirm your choice and to enter the pathname.

6.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools → Build...** to rebuild the application.

To restore the changed setting:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the *COMPILE=arm group in the List of Entries pane.

3.   Right-click on the *Tool_path setting in the Settings Values pane, and select **Reset to Empty** from the context menu.

4.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

5.   Select **Tools → Build...** to rebuild the application.

5.3 Working with source files

The examples in this section describe operations on source files:

- *Adding source files to a project*
- *Removing source files from a project* on page 5-7
- *Compiling a specific source file* on page 5-7
- *Excluding source files from a build* on page 5-8
- *Specifying source search paths* on page 5-9
- *Source path mappings* on page 5-12.

5.3.1 Adding source files to a project

To add source files to an existing project:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Right-click on the required group, in the List of Entries pane, the left pane, that defines the build stage that you want to modify, for example *COMPILE=arm, and select **Expand whole Tree** from the context menu.

   You can also expand, and collapse, groups in the List of Entries pane by clicking on the plus sign, or the minus sign, at each node in the tree. If you double-click on the group name, this expands the group and displays the contents in the Settings Values pane.

3. Select the *Sources group in the List of Entries pane to display the contents.

4. Right-click on the default Files setting, at the top of the list, and select **Edit as Filename** from the context menu.

   This displays the Enter New Filename dialog box where you can locate source files to add to the project. Select the source file you want to add and click **Save**.

   Alternatively, right-click on the default Files setting, at the top of the list, and select **Manage List...** from the context menu. Use the Settings: List Manager dialog box to modify the source list as required. You can also use this dialog box to add more files to, or to remove files from, the group. Click **OK** when you have finished adding files using the Settings: List Manager dialog box.

5. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6. Select **Tools → Build...** to rebuild the application.

You can also use this method to add files to other groups, for example other COMPILE groups, ASSEMBLE groups, or the BUILD group.

Before adding files to the active project in this way, you must wait for any build process to complete and the results to be visible in the Output pane.

―――― **Note** ――――

You do not have to add `.h` files to a project, because these are referenced from the main source files for your project. See *Specifying paths to include files* on page 5-19.

5.3.2 Removing source files from a project

To remove source files from a project:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Right-click on the required group, in the List of Entries pane, that defines the build stage that you want to modify, for example `*COMPILE=arm`, and select **Expand whole Tree** from the context menu.

3. Select the `*Sources` group in the List of Entries pane.

4. Right-click on the `*Files` setting for the source file that is to be removed, and select **Delete** from the context menu.

   The setting value for the source file is deleted.

5. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6. Select **Tools → Build...** to rebuild the application.

5.3.3 Compiling a specific source file

To compile a specific source file that is part of an existing project:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Right-click on the required group, in the List of Entries pane, that defines the build stage that you want to modify, for example `*COMPILE=arm`, and select **Expand whole Tree** from the context menu.

3. Select the `*Sources` group in the List of Entries pane.

4. Right-click on one of the source files and select **Compile File...** from the context menu. This recompiles the file and displays any debugger messages in the Output pane. It is not necessary to close the Project Properties window to view these messages.

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6.   Select **Tools → Build...** to rebuild the application.

5.3.4   Excluding source files from a build

To exclude a source file from a build:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the required group, in the List of Entries pane, that defines the build stage that you want to modify, for example ∗COMPILE=arm, and select **Expand whole Tree** from the context menu.

3.   Select the ∗Sources group in the List of Entries pane.

4.   Right-click on the ∗Files setting for the source file that is to be excluded, and select **Exclude this file from Build** from the context menu.

      The setting is grayed out and an exclamation mark (!) is added to the start of the filename.

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6.   Select **Tools → Build...** to rebuild the application.

Re-including a source file into a build

To re-include a source file into a build:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the required group, in the List of Entries pane, that defines the build stage that you want to modify, for example ∗COMPILE=arm, and select **Expand whole Tree** from the context menu.

3.   Select the ∗Sources group in the List of Entries pane.

4.   Right-click on the ∗Files setting for the source file that is currently excluded, and select **Re-Include this in Build** from the context menu.

      The setting color is restored and the exclamation mark is removed from the start of the filename.

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6. Select **Tools** → **Build...** to rebuild the application.

5.3.5 Specifying source search paths

By default RealView Debugger searches for application source file paths according to information contained in the image. If no paths are provided in the image file, RealView Debugger uses the list of directories specified in the project settings. If this fails, or where no search path has been specified, RealView Debugger looks in the current working directory for the source file or files.

In the following example, the project and the image that it builds have been created in one directory and then moved to another location for debugging. This means that RealView Debugger cannot locate source files:

1. Ensure that the Project Properties window is not open. When searching for source files, RealView Debugger updates the project properties as necessary. This fails if the window is already open.

2. Close any files that are open in the File Editor so that they are not found, and included, in the search mechanism.

3. Select **Project** → **Open Project...** and open the project, for example `dhrystone.prj`.

4. Connect to your target and load the associated image, for example `dhrystone.axf`.

5. RealView Debugger loads the image but cannot locate a source file and so displays the source search prompt, shown in Figure 5-2.
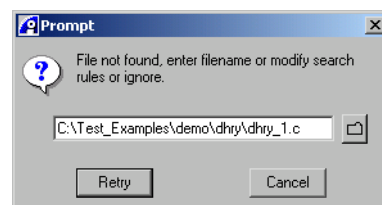
Figure 5-2 Source search prompt

If you close the prompt, for example by clicking **Cancel**, RealView Debugger warns you that it cannot locate source files for the new image. The project properties are unchanged.

You can use the source search prompt to change the search rules that RealView Debugger uses. There are two ways to do this:
*   *Autoconfiguring search rules* on page 5-10
*   *Manually configuring search rules* on page 5-11.

Autoconfiguring search rules

Use the source search prompt, shown in Figure 5-2 on page 5-9, to autoconfigure the search rules:

1.   Either:

     •   Edit the pathname shown in the prompt.

     •   Click the directory button to locate the required file.

2.   Click **Retry**.

If RealView Debugger fails to locate the file, the source search prompt remains for you to try again. If RealView Debugger succeeds, the source search prompt closes automatically.

——— **Note** ———

If the required file is already listed when you click the directory button, select it to update the project properties. The source search prompt then closes automatically.

Pathnames that you add this way automatically update the project properties for the active project. Where your project is an in-memory auto-project, RealView Debugger also saves the project settings file, for example dhrystone.axf.apr.

Select **Project** → **Project Properties...** to display the Project Properties window where you can see the new search rules. shown in the example in Figure 5-3.
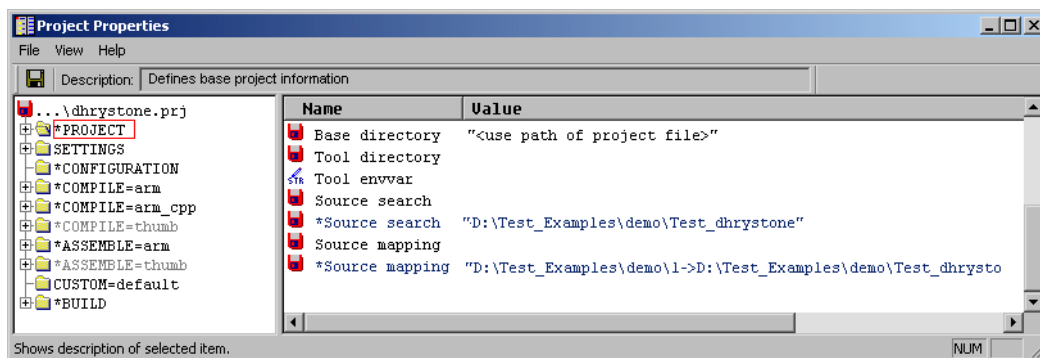
Figure 5-3 Autoconfigured search rules in the Project Properties window

As shown in Figure 5-3, absolute pathnames are used to define the new search rules. However, if you modify the source search path yourself, pathnames become relative (see *Manually configuring search rules* on page 5-11 for details).

For more details on other entries in this window see *Source path mappings* on page 5-12.

Manually configuring search rules

If you have an open project, you can specify search paths using the Project Properties window. Any search that you define is then saved with the project settings and additionally used to locate source files.

To configure the search rules manually:

1.   Select **Project → Project Properties...** to display the Project Properties window, shown in Figure 5-4.
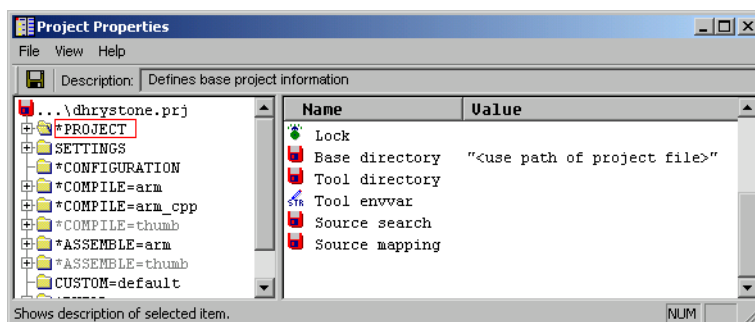
Figure 5-4 Search rules in the Project Properties window

For more details on other entries in this window see *Source path mappings* on page 5-12.

2.   Right-click on Source_search and use **Edit as Directory Name...** to specify the new location, shown in Figure 5-4.
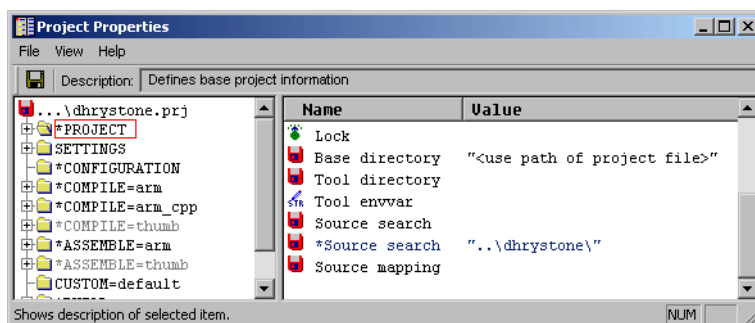
Figure 5-5 New search rules in the Project Properties window

3.   Click **File** → **Save and Close** to confirm your choice and close the Project Properties window.

Do not leave the Project Properties window open when you are working with the debugger. When searching for source files, RealView Debugger changes the project properties as necessary. This fails if the window is already open.

———— **Note** ————

As shown in Figure 5-5 on page 5-11, relative pathnames are used to define the new search rules when you configure them in this way. If you modify an autoconfigured source search path, absolute pathnames become relative.

5.3.6   Source path mappings

When you use the directory button to locate the source file, using the prompt shown in Figure 5-2 on page 5-9, RealView Debugger creates a mapping between the original file and the new location you specify. This mapping is then applied to subsequent file searches so that the debugger can locate files automatically that have the same mapping.

Pathname remappings are stored in the project settings so that they can be used in the next session, shown in the example in Figure 5-6.
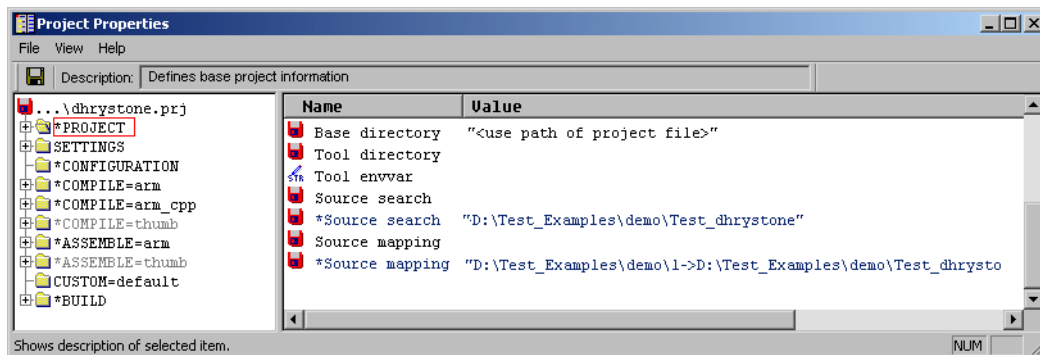
Figure 5-6 Source search paths and remappings in the Project Properties window

Figure 5-6 shows a project where:
•   a new source search path is specified
•   a source path has been remapped.

These changes mean that RealView Debugger can locate source files for the associated project.

Working with path mappings

The `Source_mapping` entry, created by editing the pathname in the source search prompt, consists of a text field containing two directories separated by ->. By substituting matching stub directories, RealView Debugger can locate missing source files. For example, suppose that your project was built using the file:

```
D:\Test_Examples\demo\ARM940T\dhrystone\dhry_1.c
```

And the project was later moved to:

```
D:\testfiles\dhrystone
```

The `Source_mapping` entry, created by editing the pathname in the source search prompt contains:

```
"D:\Test_Examples\demo\ARM940T -> D:\testfiles"
```

Remember the following when working with the `Source_search` and `Source_mapping` settings:

- You can specify a list of `Source_search` and `Source_mapping` entries where the list order defines the search order (see *Adding source files to a project* on page 5-6 for details on using the Settings: List Manager dialog box).

- You can edit `Source_mapping` settings in the Project Properties window. However, it is recommended that you use the source search prompt (shown in Figure 5-2 on page 5-9) to set up these entries.

- To remove a setting, right-click on a `Source_search` or a `Source_mapping` entry and select **Delete** from the context menu.

- Changes to source search paths, or path remappings, are reflected in the project settings for the active project. If you use the Project Control to change the active project, these new settings might not apply to the currently loaded image.

- The `Source_search` and `Source_mapping` settings are saved in the `PROJECT` group and, therefore, cannot be merged into a new user-defined project.

5.4 Working with object files

The examples in this section describe operations on object files:

- *Changing the location of object files*
- *Adding object files* on page 5-15.

5.4.1 Changing the location of object files

To change the location of object files built from project source files:

1. Select **Project** → **Project Properties...** to display the Project Properties window.

2. Select the *COMPILE=arm group in the List of Entries pane, the left pane.

3. Right-click on the Obj_location setting in the Settings Values pane, the right pane, and select **sub_dir** from the options list.

   Making this change means that object files are located in a directory called \objects inside the project base directory.

4. Right-click on the Obj_sub setting in the Settings Values pane, and select **Edit Value** from the context menu.

5. Type test_objects and press Enter to set this new destination location.

6. Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7. Select **Tools** → **Build...** to rebuild the application.

8. Use Windows Explorer to view the newly-built object files in the specified location in the project base directory.

This might be useful if you want to have different output from two projects using the same source files and same base directory. You can specify the location of the object files so that a potentially dangerous mix-up does not happen, for example when building the two applications in sequence.

You can also control object files using different CONFIGURATION groups as described in Chapter 8 *Managing Build Target Configurations*.

——— **Note** ———

Be careful here if multiple projects share the same directory or use the same source paths for output files.

———————————

5.4.2   Adding object files

The BUILD group for a project contains a default Objects setting. Use this to define object files that you want to link in to your project but which are:

• not built by this project

• not part of a library referenced from this project.

This is equivalent to listing a specific object file on the command line that invokes the linker.

To add object files to the build:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the *BUILD group in the List of Entries pane.

3.   To add a single object:

a.   Right-click on the Objects setting in the Settings Values pane, and select **Make New** from the context menu.

b.   Use in-place editing to enter the pathname of the required object file, for example, C:\Add_proj_2\Objects\thumbtest.o.

c.   Confirm the entry to see the new object file added to the list.

——— **Note** ———

You can also select **Edit as Filename** to locate the required object file.

RealView Debugger displays a *Objects entry for each object you reference.

4.   To add several object files:

a.   Right-click on the Objects setting, in the Settings Values pane, and select **Manage List...** from the context menu. This displays the Settings: List Manager dialog box.

b.   Click **Add** for each new object file to be added.

——— **Note** ———

You can also use the Settings: List Manager dialog box to sort and reorder the list, for example where object link order is important.

5.   Click **OK** to confirm the entries. The Settings: List Manager dialog box closes, and RealView Debugger updates the object files list in the Project Properties window.

6.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools** → **Build...** to rebuild the application.

Removing object files

To remove object files added to the build:

1.   To remove the file(s), either:

    •   Right-click on the *Objects setting in the Settings Values pane, and select **Delete** from the context menu.

    •   Right-click on the Objects setting, in the Settings Values pane, and select **Manage List...** from the context menu. This displays the Settings: List Manager dialog box.
        Click **Remove** for each new object file to be deleted.

2.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

3.   Select **Tools** → **Build...** to rebuild the application.

5.5   Adding library files

RealView Debugger locates ARM C and C++ libraries using the:

- default `Lib_paths` setting, in the BUILD group for a project
- appropriate environment variable, for example `RVCT21LIB` or `ARMLIB`.

If you have library files that are not in the normal library search path, then you must include the location of those library files.

The `BUILD` group for a project contains a `Libraries` setting. Use this to define the location of library files:

1.   Select **Project** → **Project Properties...** to display the Project Properties window.

2.   Select the `*BUILD` group in the List of Entries pane, the left pane.

3.   To add a single library:

   a.   Right-click on the `Libraries` setting in the Settings Values pane, the right pane, and select **Make New** from the context menu.

   b.   Use in-place editing to enter the pathname of your library files.

   c.   Confirm the entry to see the new library file added to the list, for example, `C:\Add_proj_2\Libraries\lib_proj_1.a`.

   ——— **Note** ———

   You can also select **Edit as Filename** to locate your library files.

   ————————————————

   RealView Debugger displays a `*Libraries` entry for each library you reference.

4.   To add several library files:

   a.   Right-click on the appropriate `Libraries` setting, in the Settings Values pane, and select **Manage List...** from the context menu. This displays the Settings: List Manager dialog box.

   b.   Click **Add** for each new library file to be added, enter the path and filename for the library file, then click **Add** to add it to the list.

   ——— **Note** ———

   You can also use the Settings: List Manager dialog box to sort and reorder the list, for example where link order is important.

   ————————————————

5.   Click **OK** to confirm the entries. The Settings: List Manager dialog box closes, and the library paths list in the Project Properties window is updated.

6.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools** → **Build...** to rebuild the application.

5.5.1   Removing library files

To remove library files added to the build:

1.   Remove the file(s), either:

   - Right-click on the ✳Libraries setting in the Settings Values pane, and select **Delete** from the context menu.

   - Right-click on the Libraries setting, in the Settings Values pane, and select **Manage List...** from the context menu. This displays the Settings: List Manager dialog box.
     Click **Remove** for each new file to be deleted.

2.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

3.   Select **Tools** → **Build...** to rebuild the application.

 ARM DUI 0291C

5.6   Specifying paths to include files

For C and C++ sources, if you have project-specific header files and these are in a different directory from the rest of your sources, you must specify the path to include the files. For assembler sources, you can specify source file search paths.

To specify paths to include files that are in different directories to your main source files:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the appropriate *COMPILE or *ASSEMBLE group in the List of Entries pane, the left pane, and select **Expand whole Tree** from the context menu.

3.   Select *Preprocessor in the List of Entries pane, the left pane.

4.   Right-click on Include in the Settings Values pane, the right pane, and select **Edit as Directory Name** from the context menu. The Enter New Directory: dialog box is displayed.

5.   Double-click to open the required directory, then click **Select**.

6.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools → Build...** to rebuild the application.

5.7 Configuring linker options

The examples in this section describe operations to specify how files are linked:

- *Specifying linker options and scatter loading*
- *Adding prelink and postlink commands*.

5.7.1 Specifying linker options and scatter loading

The default output from the ARM linker is a non-relocatable image where the code starts at `0x8000` and the data section is placed immediately after the code. You can specify exactly where the code and data sections are located by using linker options or a scatter-loading description file.

See the documentation for your build tools for more details on the ARM linker options. For example, if you are using *RealView Developer Kit* (RVDK), see *RealView Developer Kit Linker and Utilities Guide*.

To specify scatter loading:

1. Select **Project** → **Project Properties...** to display the Project Properties window.

2. Right-click on the `*BUILD` group in the List of Entries pane, the left pane, and select **Expand whole Tree** from the context menu.

3. Select the `Link_Advanced` group in the List of Entries pane.

4. Right-click on `Scatter_file` in the Settings Values pane, the right pane, and select **Edit as Filename** from the context menu to locate a previously created scatter-loading description file for the project image (see your build tools documentation for details on creating scatter-loading description files). Alternatively, specify other linker options as required.

5. Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6. Select **Tools** → **Build...** to rebuild the application.

5.7.2 Adding prelink and postlink commands

Prelink and postlink makefile commands are run only when the linker is run. Prelink commands can be used to copy libraries or objects from other developers so that you do not have to build them yourself. Postlink commands might be used to convert the executable file into another format, for example to load to Flash or for loading by a specific operating system.

To add prelink and postlink commands to the build:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the *BUILD group in the List of Entries pane, the left pane, and select **Expand whole Tree** from the context menu.

3.   Select the Pre_Post_Link group in the List of Entries pane.

4.   Right-click on the Pre_link setting in the Settings Values pane, the right pane, and select **Edit Value** from the context menu.

5.   Type @+echo This is before linkage and then press Enter to confirm the value.

        This string is output every time you build or rebuild, the application, but the command is not displayed when it is executed.

6.   Right-click on the Post_link setting in the Settings Values pane, and select **Edit Value** from the context menu.

7.   Type +copy $(PROGRAM) $\temp.axf and then press Enter to confirm the value.

        Type this exactly as shown. Your window looks like Figure 5-7.
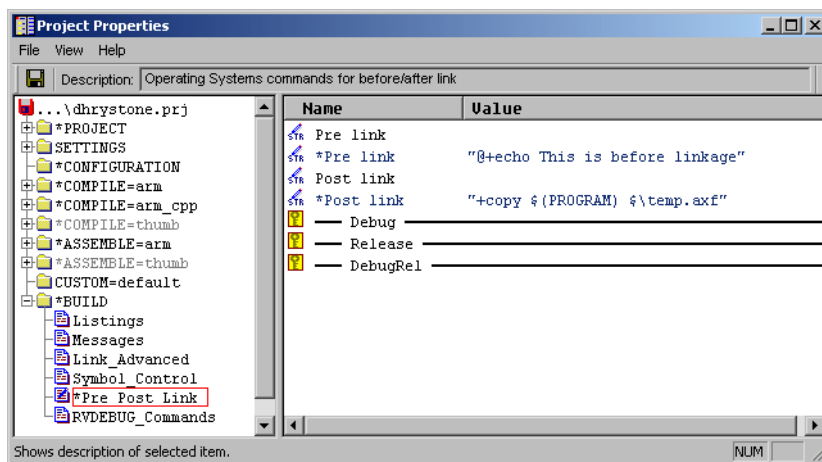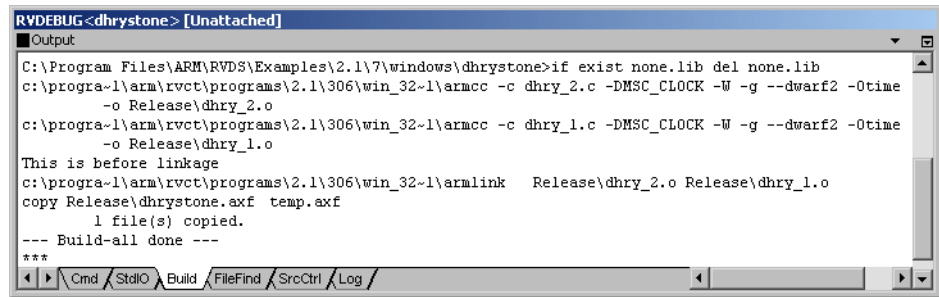
Figure 5-7 Linker options in the Project Properties window

This postlink command copies the output from the build, that is dhrystone.axf, to another location. In this example, you are sending the executable file temp.axf to the current working directory, but it might be a predefined central location for images to be debugged.

The copy command is displayed in the **Build** tab when executed.

8. Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

9. Select **Tools** → **Build...** to rebuild the application.

10. Click on the **Build** tab, if necessary, to see the linker output, shown in Figure 5-8.


```
RVDEBUG<dhrystone> [Unattached]
Output                                                                          ▾  ☑
C:\Program Files\ARM\RVDS\Examples\2.1\7\windows\dhrystone>if exist none.lib del none.lib
c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armcc -c dhry_2.c -DMSC_CLOCK -W -g --dwarf2 -Otime
        -o Release\dhry_2.o
c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armcc -c dhry_1.c -DMSC_CLOCK -W -g --dwarf2 -Otime
        -o Release\dhry_1.o
This is before linkage
c:\progra~1\arm\rvct\programs\2.1\306\win_32~1\armlink   Release\dhry_2.o Release\dhry_1.o
copy Release\dhrystone.axf  temp.axf
        1 file(s) copied.
--- Build-all done ---
***
◀ ▶ \ Cmd ⟨ StdIO ⟩ Build ⟨ FileFind ⟨ SrcCtrl ⟨ Log /          ◀         ▶ ▾
```

Figure 5-8 Linker output in the Output pane

——— **Note** ———

Makefile commands can include operating system commands.

5.8   Specifying breakpoints

Projects enable you to control an application that you are debugging. You can include image load commands and set breakpoints that are stored as part of the project. The SETTINGS group contains two groups for breakpoints:

Auto_Set_Breaks

> Set as soon as a symbol is matched. If there is no symbol specified, these are set on any load. These breakpoints appear in the Break/Tracepoints pane in the usual way.

Named_Breaks

> Breakpoints that you want to set often. Enables you to set breakpoints that are specific to the application, to an RTOS, or to a library.

──────── Note ────────

The steps for adding Auto_Set_Breaks and Named_Breaks are identical, except that for Auto_Set_Breaks you can choose to have RealView Debugger prompt you before setting the breakpoint. The steps in this section describe setting Named_Breaks.

────────────────────────

To add named breakpoints:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the *SETTINGS group in the List of Entries pane, the left pane.

3.   Right-click on the Named_Breaks group in the Settings Values pane, the right pane, and select **Explore** from the context menu.

4.   Right-click on the Default group in the Settings Values pane, and select **Make Copy...** from the context menu.

5.   Enter a new name for the group, for example My_breakpoints, and click **Create**.

6.   Right-click on the My_breakpoints group in the Settings Values pane, and select **Explore** from the context menu.

7.   Right-click on the Cmd setting, in the Settings Values pane, and select **Edit Value** from the context menu.

8.   Type bi \DHRY_1\#149:5 and then press Enter to confirm the value.

     This identifies a software breakpoint on the chosen instruction.

9.   Right-click on the Description setting, in the Settings Values pane, and select **Edit Value** from the context men.

The text entered here appears in the list selection box to identify the named breakpoint.

10. Type My test breakpoint and then press Enter to confirm the value.

    This text identifies the named breakpoint.

11. Create a new group, for example My_test_breakpoints, and set up a second named breakpoint if required.

12. Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

13. Select **Tools** → **Build...** to rebuild the application.

14. Connect to your target and load the newly-built image dhrystone.axf.

15. Click on the **Src** tab to view the source file dhry_1.c.

16. Select **Debug** → **Simple Breakpoints** → **Named...** to display the list selection box, shown in Figure 5-9.
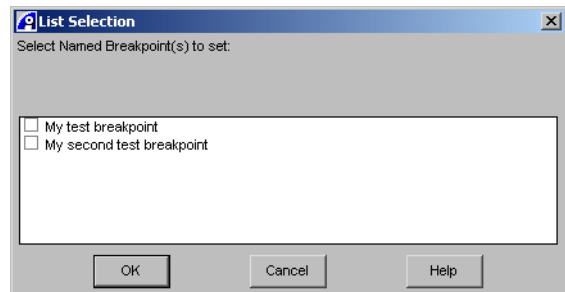
Figure 5-9 Named Breakpoints selection box

This box lists the named breakpoints you previously set up for the project. Select the breakpoints you want to set, then click **OK**. These breakpoints are also available when you next open the project.

When you select breakpoints from the Named Breakpoints list, they appear in the Break/Tracepoints pane in the usual way.

———— **Note** ————

Specifying breakpoints in this way can also be applied to an auto-project.

———————————————

5.9 Customizing the build

The example in this section describes how to customize a build:

- *Adding a custom build group*
- *Running the custom build* on page 5-27.

5.9.1 Adding a custom build group

In this example, you add a new CUSTOM group to the project build model and then configure this group to display a message during the build. Custom groups can be created at different points in the settings hierarchy and easily disabled when not required.

There are two ways to configure a CUSTOM group, either:

- Create a new CUSTOM group in a specified location in the hierarchy.
- Copy the default CUSTOM=default group for the project and then rename this group.

This example creates a new CUSTOM group so that default settings are shown. Where you have not made any changes to the CUSTOM=default group for the project, this gives the same result.

To set up the new build group:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Right-click on a group in the List of Entries pane, the left pane.

    Where you click defines the location of the new group in the hierarchy, for example, right-click on the *COMPILE=arm group to see the new entry inserted below this group.

3. Select **Make New...** from the context menu to display the Group Type/Name selector dialog box.

4. Highlight CUSTOM in the Group Type display list.

5. Type MY_GROUP in the Group Name field.

6. Click **OK** to close the Group Type/Name selector dialog box.

    The contents of the new group are displayed in the Settings Values pane, the right pane, of the Project Properties window.

7. Right-click on the Message setting, in the Settings Values pane, and select **Edit Value** from the context menu.

8. Type Writing version to version file and then press Enter to confirm the value.

This string is output every time you build or rebuild the application.

9. Right-click on the Files setting, in the Settings Values pane, and select **Edit Value** from the context menu.

10. Type version.txt and then press Enter to confirm the value.

11. Right-click on the Depends_on setting, in the Settings Values pane, and select **Edit Value** from the context menu.

12. Type $(MAKENAME) and then press Enter to confirm the value. Ensure that you do not include any spaces in the entry.

    This entry means that the output file is dependent on the makefile so you rebuild version.txt each time the makefile is updated. Usually, dependent files are inputs to a build but this example illustrates the method.

    Do not make the executable file, dhrystone.axf, the dependent because that does not work. Instead, you must use postlink commands as shown in *Adding prelink and postlink commands* on page 5-20.

13. Right-click on the Command setting, in the Settings Values pane, and select **Edit Value** from the context menu.

14. Type +echo 'version = 1.00 >$@' and then press Enter to confirm the value.

    This command writes the message to the text file defined previously. You might use a command to run another program or to output the date.

    Preceding the command with a plus sign means that the command is a built-in operating system shell command. Do not put a plus sign in the command when running normal applications. If you do not want the command to be shown while it is running, put an at sign (@) in front of the command string.

    The Project Properties window looks like Figure 5-10 on page 5-27.

 ARM DUI 0291C
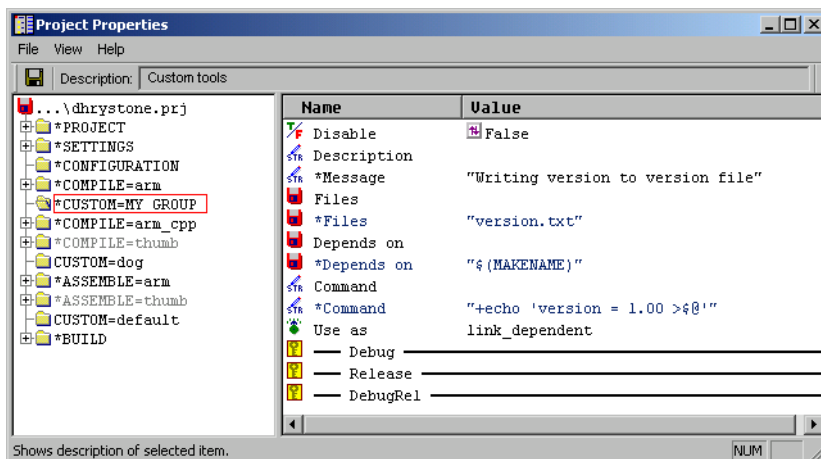
Figure 5-10 Customizing the build

15. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

16. Select **Tools → Build...** to rebuild the application.

17. Click on the **Build** tab, if necessary, to see the build output.

In this example, you are sending the output file version.txt to the current working directory. Open the messages file in the File Editor pane and view the contents.

5.9.2   Running the custom build

You can control the build using the Use_as settings value, shown in Figure 5-10. Right-click on the setting to see the available options:

link_dependent

> RealView Debugger ensures that it is up-to-date when building the application. This is the default.

link_input   Creates an object file or library which is then linked to the application built by the project.

named_target

> Makes the file a named target. This means that other files must be dependent on it for it to be used.

Use this option to create a header or source file, that is a `.c` or `.asm` file, that is then compiled or assembled in a `COMPILE` or `ASSEMBLE` group. The `make` utility knows to build the header or source file before building the application with it.

You can also layer custom projects in this way with the output of one project used as the input of another.

post_link   Specifies that the file is built only after linking.

——— **Note** ———

You can force a `CUSTOM` rule to run every time that you build the application by referring to a file that does not exist. For example, in the steps in *Adding a custom build group* on page 5-25, you could have specified that the file was called `version` but still written to `version.txt`. This causes the `CUSTOM` rule to be executed each time you build the application because the `make` utility decides that the file called `version` does not exist and so tries to build it.

To undo this change and restore the build model:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the `*CUSTOM=MY_GROUP` group in the List of Entries pane, and select **Delete** from the context menu.

3.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

4.   Select **Tools → Build...** to rebuild the application.

5.10   Interworking ARM and Thumb

You can mix C and C++ code for ARM and Thumb®, provided that the code conforms to the requirements of the Procedure Call Standard for the ARM Architecture.

The ARM linker detects when ARM and Thumb code is mixed and generates small code segments, called *veneers*, that control the change in state between ARM and Thumb.

If you compile a module for interworking, it generates slightly larger code for Thumb, and marginally larger code for ARM. Use the linker option `--info veneers` to find the amount of space taken up by the veneers. Disabled by default, this can be set in the `BUILD` group for your interworking project.

---------- **Note** ----------

ARM code compiled for interworking cannot be used on ARM® processors that are not Thumb-capable, for example StrongARM®, because these processors do not implement the `BX` (Branch Exchange) instruction.

This section contains examples of making changes to the project `armtothumb.prj` located in *install_directory*\RVDK\Examples\...\interwork\ . You might want to make a backup of the project base directory before following the examples so that the default files and settings can be restored.

The examples in this section describe:

•   *Running the interworking example*
•   *Setting compiler options for interworking* on page 5-30
•   *Displaying code sizes* on page 5-32.

5.10.1   Running the interworking example

To see an example of interworking ARM and Thumb code:

1.   Select **Project → Open Project...** from the default Code window main menu to open the project file `armtothumb.prj` located in the directory:

         install_directory\RVDK\Examples\...\interwork\

2.   Open the source file `thumbsub.c` so that it is displayed in the File Editor pane.

3.   Select **Edit → Editing Controls → Show Line Numbers** to display line numbers.

         This is not necessary but might help you to follow the example.

4.   Click on the file tab to view the source file `thumbsub.c`.

---

5. Add a new line immediately before the printf statement, for example:

   5 fprintf(stdout,"Now in Thumb routine\n");

   6 printf("Hello and goodbye from Thumb\n");

6. Select **Tools → Build...** to rebuild the application.

   This displays a list selection box where you can confirm the source file that has been changed.

7. Click **OK** to confirm the save and rebuild the application.

   If this is the first time you have built the example project, accept the option to create the makefile.

8. Connect to a Thumb-capable debug target using RVI or RVI-ME.

9. Click on the **Src** tab to see the hyperlink to load the image associated with the open project.

10. Click on the hyperlink to load the image armtothumb.axf.

11. Click **Go** to run the application.

    The **StdIO** tab, in the Output pane, displays the results.

5.10.2   Setting compiler options for interworking

To set compiler options:

1. Select **Project → Project Properties...** to display the Project Properties window. Expand the entries, shown in Figure 5-11 on page 5-31.
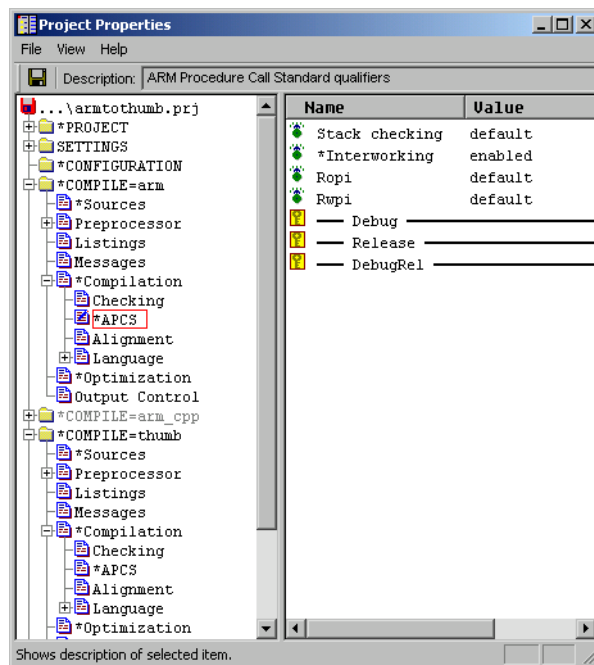
Figure 5-11 Project Properties window for armtothumb project

2. Confirm the compiler options for the armtothumb example project.

The *Interworking settings value is set to enabled for the *COMPILE=arm group.
This sets the --apcs /interwork compiler option. This means that the ARM
compilers can compile modules containing routines called by other routines
compiled for Thumb state.

Similarly, expand the *COMPILE=thumb group to see the APCS setting enabled for
routines compiled for Thumb state.

3. The armtothumb example project defines two enabled COMPILE groups, shown in
Figure 5-11:

- COMPILE=arm specifies the ARM C compiler, to compile ISO C source into
  32-bit ARM code

- COMPILE=thumb specifies the Thumb C compiler to compile ISO C source
  into 16-bit Thumb code.

The COMPILE=arm_cpp settings are not used for this project. This specifies the ARM
C++ compiler, to compile ISO C++ source into 32-bit ARM code.

4.   Ensure that all ARM and Thumb modules are compiled to the same standard if they are to be interworked. Failure to do this results in a warning from the compiler, for example:

```
Error: L6239E: Cannot call non-interworking THUMB symbol 'thumb_function'
in thumbsub.o from ARM code in armmain.o(.text)
```

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6.   Select **Tools → Build...** to rebuild the application.

See the documentation for your build tools for more details on the `--apcs` compiler option and its qualifiers. For example, see *RealView Developer Kit Compiler and Libraries Guide*.

5.10.3   Displaying code sizes

To display the code sizes for your interworking project:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Right-click on the `*BUILD` group at the bottom of the List of Entries pane, the left pane, and select **Expand whole Tree** from the context menu. This displays the contents in the Settings Values pane, the right pane.

3.   Select the `Messages` group in the List of Entries pane.

4.   Right-click on the `Sizes` setting, in the Settings Values pane, and select **both** from the options list. This displays details and totals size information.

5.   Right-click on the `Veneers` setting, in the Settings Values pane, and select **enabled** from the options list.

6.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools → Build...** to rebuild the application.

8.   The sizes are displayed in the **Build** tab of the Output pane.

See the documentation for your build tools for more details on veneers, code size, and linker options. For example, if you are using RVDK, see *RealView Developer Kit Linker and Utilities Guide*.

5.11   Using MS-DOS applications

The examples in this section describe operations on source files:

- *Configuring tools support*
- *Using MS-DOS names under Windows* on page 5-34.

5.11.1   Configuring tools support

When running older 16-bit tools, you might see a range of symptoms during different stages of a build, for example slow echoing of keystrokes, disk access errors, or rogue `Winoldapp` tasks left in the system. You can set RealView Debugger to support these older tools that use extended memory, and run your old tools in a DOS dialog box.

——— **Note** ———

Do not set this option for new 32-bit applications because this severely impacts performance.

To change this behavior:

1. Select **Project → Build-Tool Properties** to display the Build-Tool Properties window.

2. Select the `*PROC=` group corresponding to your build tools in the List of Entries pane, the left pane.

3. Right-click on the `Dos_app` setting at the bottom of the Settings Values pane, the right pane, and select **True** from the context menu.

4. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Build-Tool Properties window.

5. Click **OK** at the prompt to regenerate the project because of the change to the toolchain.

6. Select **Tools → Build...** to rebuild the application.

5.11.2 Using MS-DOS names under Windows

You can amend your project properties to accommodate tools that cannot handle:

- long filenames, that is, names greater than eight characters
- spaces in filenames.

To change this behavior:

1. Select **Project → Build-Tool Properties** to display the Build-Tool Properties window.

2. Select the `*PROC=` group corresponding to your build tools in the List of Entries pane, the left pane.

3. Right-click on the `Dos_names` setting at the bottom of the Settings Values pane, the right pane, and select **always** from the options list.

4. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Build-Tool Properties window.

5. Click **OK** at the prompt to regenerate the project because of the change to the toolchain.

6. Select **Tools → Build...** to rebuild the application.

Chapter 6
Managing Projects in the Code Window

This chapter explains how to use the RealView® Debugger GUI to manage your projects. It describes how to work with projects using the Project Control dialog box and the Process Control pane. It contains the following sections:

- *Using the Project Control dialog box* on page 6-2
- *Managing projects in the Process Control pane* on page 6-5.

6.1   Using the Project Control dialog box

You can use the **Project** and **Tools** menus to perform operations on your open projects. By default, these menus give you access to the active project. For more details on these menus see:

- *Using the Project menu* on page 2-2
- *Using the Tools menu* on page 7-2.

Where you are working with multiple projects, you can use the Project Control dialog box to act on other projects or to change the project environment.

The Project Control dialog box enables you to:

- control project binding
- display the Project Properties window for a project
- perform build, rebuild, or clean actions on a project
- change the active project
- upgrade a project to use a new toolchain.

This section describes:

- *Viewing the Project Control dialog box*
- *Working with the Project Control dialog box* on page 6-3.

6.1.1   Viewing the Project Control dialog box

Select **Project** → **Project Control...** from the default Code window main menu to display the Project Control dialog box, shown in Figure 6-1.
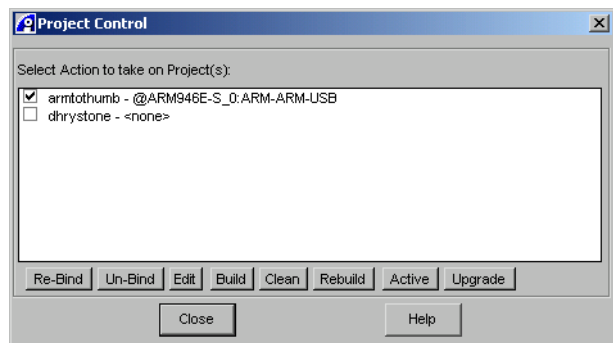
Figure 6-1 Project Control dialog box

The Project Control dialog box displays the *open project list*. When RealView Debugger opens a user-defined project, or an auto-project, it is added to the top of the open project list.

The open project list shows the order in which the projects were opened. If you are not connected when you open projects, the most recent project, the *default active project*, is at the top of the list. This is selected by default. If you are connected, the projects are in the same order but the last project to open is not necessarily the active project, see *Working with multiple projects* for details.

Working with multiple projects

If you are connected and you open multiple projects, the open project list shows which project is bound to the current connection. In Figure 6-1 on page 6-2, there are two projects in the open project list. The last project to open is shown at the top of the list and is selected by default (networking). The primes project is bound to the RVI-ME connection, @ARM946E-S_0:ARM-ARM-USB [Unattached], using default binding. The first project to open (dhrystone) is shown at the bottom of the list. In this case, primes is the active project because it was the last project to bind successfully to the current connection.

The Project Control dialog box enables you to carry out operations on all open projects. In some cases, you can complete actions on several projects, for example building, but other actions can only be carried out on a single project, for example using the **Edit** button to view the project settings file.

If you select multiple projects from the list and then click an action control, the specified action is carried out on the selected projects in the order they are listed and so, in some cases, the last action completed successfully overwrites all previous actions.

6.1.2   Working with the Project Control dialog box

To perform an action on a project:

1.   Select **Project → Project Control...** from the default Code window main menu to display the Project Control dialog box.

2.   Click on a project in the list so that the associated check box is checked. This selects a project for action.

3.   Use the controls in the dialog box to operate on the chosen project:

   Re-Bind   Enables you to bind a project to a specific connection. This is most useful when you are working with multiple projects and multiple connections. See *Forcing binding* on page 9-9 for details.

   If successful, changes to project binding are reflected in the **Process** tab in the Process Control pane, if this is visible.

Un-Bind Enables you to unbind a project from a specific connection. This is most useful when you are working with multiple projects and multiple connections. See *Forcing binding* on page 9-9 for details.

        If successful, changes to project binding are reflected in the **Process** tab of the Process Control pane, if this is visible.

Edit Displays the Project Properties window for the chosen project ready for editing. This is independent of the active project.

Build Enables you to build one or more applications. If one build fails, RealView Debugger asks for confirmation before continuing with the other builds listed. Applications are built in the order shown in the display list.

Clean Enables you to clean one or more projects. Projects are cleansed in the order shown in the display list.

Rebuild Enables you to clean and rebuild one or more applications. If one rebuild fails, RealView Debugger asks for confirmation before continuing with the other rebuilds. Applications are rebuilt in the order shown in the display list.

Active Makes the chosen project the active project if you have more than one project open. The active project is reflected in the title bar in the default Code window that changes to show the new active project. The project binding remains unchanged. See *Active projects* on page 10-4 for more details.

Upgrade Displays the Upgrade Project dialog box, where you can upgrade the project to use a new toolchain. If you try to upgrade a project where no upgrades exist, RealView Debugger displays a message box to say that no upgrades are available. See *Upgrading the project toolchain* on page 2-12 for more details.

Help Displays the online **Help** menu.

Close Closes the Project Control dialog box.

6.2 Managing projects in the Process Control pane

If you are connected to a debug target, use the Process Control pane to see details about the current process and target processor. If you open a project that binds to the current connection, the Process Control pane also enables you to view your project properties and perform operations on the project files.

This section describes:

*   *Displaying the Process Control pane*
*   *Working with user-defined projects* on page 6-6
*   *Working with auto-projects* on page 6-7
*   *Project context menus* on page 6-8.

6.2.1 Displaying the Process Control pane

To display the Process Control pane, shown in Figure 6-2, select **View** → **Process Control** from the Code window main menu.
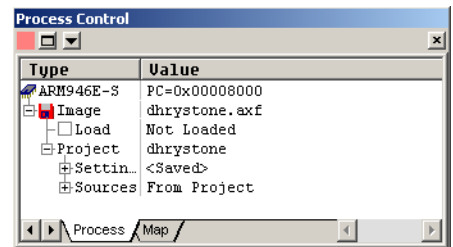
Figure 6-2 Process tab in the Process Control pane

In Figure 6-2 the:

*   Process Control pane is floating and so the title bar reflects the calling Code window

*   **Process** tab shows details about the target processor

*   image defined by the active project, dhrystone.axf, is not loaded

*   source files are being collected from the project (and not from the image)

*   user-defined project dhrystone is bound to the connection.

——— **Note** ———

The Settings entry for user-defined projects is always set to <Saved>.

Navigating the Process Control pane

When working with entries in the Process Control pane, you can use type ahead facilities to locate files. This is especially useful where your project specifies a large number of source files. For example, type the first letter of the source file that you want to view. RealView Debugger expands the Sources entry and locates the first matching occurrence. When using this feature, the type ahead buffer is case insensitive and is limited to 128 characters. Do one of the following to clear the buffer:

- select a different item
- press Home to move to the top of the pane
- press Escape.

6.2.2 Working with user-defined projects

When you create or open a Standard or Custom user-defined project, the project name is shown in the Project entry in the Process Control pane. The example in Figure 6-3 shows the contents for the example dhrystone project.
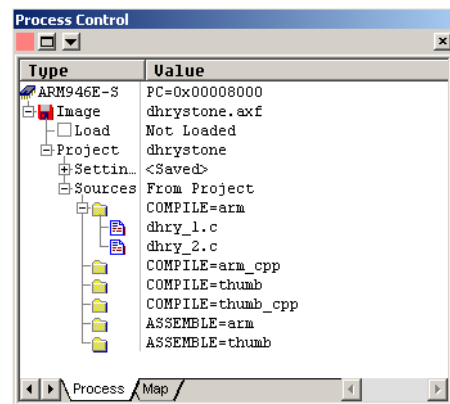
Figure 6-3 User-defined projects in the Process Control pane

The groups shown in the Process Control pane correspond to groups in the Project Properties window, see Figure 4-1 on page 4-2. All project groups are shown, that is groups that are disabled in the Project Properties window are visible in the Process Control pane. However, these groups do not affect the build model.

6.2.3 Working with auto-projects

When you load an image directly to a debug target, RealView Debugger checks to see if an auto-project exists for the image in the same location. Where an auto-project exists, RealView Debugger opens the settings file and then uses it to load the specified image. Where no auto-project exists, RealView Debugger creates an in-memory auto-project to use in this session.

——— **Note** ———

If you have created a project, it is recommended that you open this first to load and debug the associated image. This enables you to build your application and to change project settings.

Figure 6-4 shows a Process Control pane where the image dhrystone.axf has been loaded to the debug target without opening the project first.
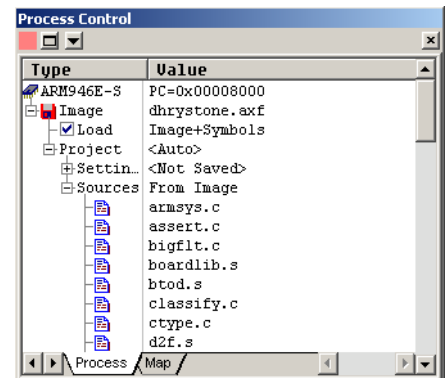
Figure 6-4 Auto-projects in the Process Control pane

In this example, the Project <Auto> entry shows that RealView Debugger is using an auto-project. An in-memory auto-project is identified by the Settings <Not Saved> entry.

When you save the settings for an auto-project, they are saved in a .apr file, for example, dhrystone.axf.apr. An auto-project that has a saved settings file is identified by a Settings <Saved> entry in the Process Control pane.

6.2.4   Project context menus

You can perform project-level operations using context menus from the Process Control pane. Right-click on the `Project`, `Settings`, or `Sources` entry, shown in Figure 6-4 on page 6-7, to display the context menus. For details on these operations see:

- *Build operations*
- *Project and Settings operations* on page 6-9
- *Source file operations* on page 6-10
- *Working on a source file* on page 6-10.

———— **Note** ————

If you have several projects open, any unbound projects are usually inaccessible from the Process Control pane by definition. See Chapter 9 *Project Binding* for details on how to work on these projects.

—————————————

Build operations

These operations are available on all context menus in the Process Control pane, unless you select a no-build project or an auto-project:

Build   Builds the application.

Rebuild All   Rebuilds the application.

Clean   Cleans the project.

Update Dependencies

Updates all dependencies for the project makefile.

Properties   Displays a text box showing information about the project. The box details a subset of the `PROJECT` group defined at creation.

Some of these operations are also available from the **Tools** menu. See *Using the Tools menu* on page 7-2 for details.

Project and Settings operations

These options are available, in addition to the build operations, when you right-click on the Project or Settings entry in the Process Control pane:

Close   Closes the project. The action depends on whether the project is a user-defined project or an auto-project:

• For a user-defined project, if the image is loaded when you close the project, you are prompted to unload the image. If you choose not to unload the image, RealView Debugger closes your user-defined project and creates an in-memory auto-project for the image. Otherwise, RealView Debugger unloads the image and closes the project.

• For an auto-project, RealView Debugger closes the project even if the image is loaded.

If the Project Properties window is visible, and you have any unsaved changes, RealView Debugger gives you the option to save the changes before the project closes.

Closing a project automatically unbinds it from the connection.

Save   Saves the in-memory settings for the auto-project to a file called *image_name*.axf.apr, and places it in the same location as the image file. This option is available only for in-memory auto-projects, that is where there is no auto-project settings file and the Settings entry shows <Not Saved>.

Delete Auto-Project File

Deletes the settings file, *image_name*.axf.apr, for the auto-project. The Settings status in the Process Control pane changes to <Not Saved>. Delete the auto-project settings file to force RealView Debugger to create a new in-memory auto-project when you next load the image.

This option is only available for a saved auto-project.

Project Properties...

Displays the Project Properties window that enables you to edit the settings for the project.

Source file operations

The Process Control pane displays a list of source files for the current project. Right-click on the Sources entry to specify how RealView Debugger collects this list to populate this pane. The operations available, in addition to the build operations, depend on the type of project:

Collect from Project

> Where selected, this indicates that the list of source files shown is derived from the project. The sources are listed in the appropriate COMPILE and ASSEMBLE groups.
>
> This is selected by default for a Standard project.

Collect from Makefile

> Select this option if you want to use a project makefile to define which source files are shown.
>
> This is selected by default for a Custom project.

Collect from Image

> Select this option if you want to use the image to define which source files are shown. The image must be loaded for the sources to be shown.
>
> This is the only option for an auto-project.

> ——— **Note** ———
>
> Files might be listed that are not included in the list of files for your project. This is because the file list comes from the image, and so the source paths might be unknown, or unavailable. For example, standard C or C++ library files might be listed.
>
> ————————————
>
> You must load the image before you change the way that source files are collected if you want to collect from the image.

Working on a source file

You can work on a source file directly from the Process Control pane. Right-click on a file and then choose the required option from the context menu. The options available depend on how the debugger collects the list of files (see *Source file operations*):

Open File   Opens a new tab in the File Editor pane in the default Code window and displays the chosen file for editing. If the selected file is already in the File Editor pane, then the corresponding tab is brought to the top for editing.

             ARM DUI 0291C

	This option is available when sources are collected from either the project or makefile.

Scope to File   RealView Debugger uses *scope* to determine the value of a symbol. Select this option to scope to the chosen file and so determine the execution *context* and define how local variables are accessed.

See the chapter describing controlling execution in *RealView Developer Kit v2.2 Debugger User Guide* for full details on scope.

This option is available only when sources are collected from the image.

Properties   Displays a text box showing details about the source file.

You can double-click on any source file in the Process Control pane to open a new tab in the File Editor pane and display the chosen file for editing. If the selected file is already in the File Editor pane, then the corresponding tab is brought to the top. RealView Debugger also scopes to the chosen file, if debug information is available.

Chapter 7
Building Applications

This chapter covers in detail the build process and how to customize it. It contains the following sections:

- *Using the Tools menu* on page 7-2
- *Managing your build tools* on page 7-5
- *Using the Build-Tool Properties window* on page 7-6
- *Defining build tools* on page 7-7
- *Building files and images* on page 7-8
- *Using the Build dialog box* on page 7-11
- *Finding build errors* on page 7-13
- *Using a build model* on page 7-14.

7.1 Using the Tools menu

The Code Window **Tools** menu contains commands that control your build processes and provide assistance when locating and correcting errors during application building.

This section describes how to use the **Tools** menu, and associated buttons, with one or more projects and forms the basis for the rest of this chapter. This section contains:

- *Active project*
- *Tools menu build options*
- *Other Tools menu options* on page 7-4
- *Using the Actions toolbar* on page 7-4.

7.1.1 Active project

RealView® Debugger enables you to have several projects open at any time. If you select a command from the **Tools** menu, RealView Debugger applies the chosen operation to the active project.

In a debugging session, the active project is usually the last project you open. If you are connected to a debug target, then the active project is the last project that RealView Debugger opens and successfully binds to the connection. The name of the active project appears in the default Code window title bar.

―――― **Note** ――――

When working with multiple projects or Container projects, you might have to change the active project so that you can access the project properties, and perform project-level operations, from the Code window. See *Changing the active project* on page 10-4 for details on how to do this.

―――――――――――

7.1.2 Tools menu build options

The **Tools** menu offers the following build options:

Build...   Builds an executable image or library.

Select this option without an open user-defined project, or where the active project is an auto-project, to display the Build dialog box. This is populated with image settings if they are available.

You can also rebuild executable or library files from RealView Debugger without making a connection using this option.

See *Using the Build dialog box* on page 7-11 for details.

Build This File

Creates an object file from the current source file selected in the File Editor pane. With a user-defined project open, the stored build model is used to perform the rebuild for the project.

If you select this option without an open project, RealView Debugger displays the Build dialog box populated with object settings.

You can also use this option to rebuild a source file from RealView Debugger without making a connection.

This option is enabled when a file is selected in the File Editor pane.

See *Using the Build dialog box* on page 7-11 for details.

Next Line/Error

If rebuilding generates a list of errors, this option enables you to move through the errors, make corrections, and check your code.

See *Finding build errors* on page 7-13 for details.

This option is also used when searching for text in files as part of your development or debugging session. Select this option to move to the next matching occurrence of the search string or expression as displayed in the **FileFind** tab of the Output pane. See Chapter 12 *Searching and Replacing Text* for details.

Stop Build/Find

Stops any build, rebuild, or search operation in progress.

Clean (remove objects)

Removes object and executable files for a project. You can use this to remove any files built by the project.

This option is enabled when a user-defined project is open that contains a makefile.

Rebuild All (Clean+Build)

Removes object, executable, or library files for a project, and then rebuilds the project using the stored build model. Use this option to force a build from scratch.

This option is enabled when a user-defined project is open that contains a makefile.

7.1.3 Other Tools menu options

Other **Tools** menu options are available, and are described in detail in other parts of the RealView Debugger documentation:

Analyzer/Trace Control

Displays options to configure and display analyzer and trace information, and to connect to a logic analyzer.

Simulation Control

The **Simulation Control** option is not available in this release.

Workspace Options...

Displays the Workspace Options window. See the chapter describing workspaces in *RealView Developer Kit v2.2 Debugger User Guide* for details on configuration options.

Options... Displays the Options window, where you specify global configuration options. See the chapter describing workspaces in *RealView Developer Kit v2.2 Debugger User Guide* for details on configuring global options.

7.1.4 Using the Actions toolbar

The Actions toolbar contains buttons that replicate selected build options from the **Tools** menu:

Build Rebuilds an executable image or library file. See the description of the **Build...** option in *Tools menu build options* on page 7-2.

Compile Creates an object file from the current source file selected in the File Editor pane. See the description of the **Build This File** option in *Tools menu build options* on page 7-2.

Stop Compile/Build/Find

Stops any build, rebuild, or search operation that is in progress. See the description of the **Stop Build/Find** option in *Tools menu build options* on page 7-2.

7.2   Managing your build tools

When you install RealView Debugger for the first time, it determines the location of your build tools to use for all user-defined projects that you create. However, you can override this setting or specify a different tool for selected projects.

——— **Note** ———

Auto-projects do not contain a build model and so make no use of build tools. This section applies to user-defined projects only.

If you have more than one version of the ARM® build tools, for example *RealView Developer Kit* (RVDK) and *ARM Developer Suite*™ (ADS), RealView Debugger uses the latest version.

——— **Note** ———

If you have upgraded your build tools since you created a project, RealView Debugger prompts you to upgrade when you next open that project. See *Upgrading the project toolchain* on page 2-12 for more details.

If you are not using a project but you want to rebuild a particular file, or set of files, RealView Debugger gives you the option to use the default build tools or to specify your own makefile or build script.

The rest of this chapter describes how to access your build tools and manage your application building.

7.3   Using the Build-Tool Properties window

Select **Project → Build-Tool Properties...** from the Code window to display the Build-Tool Properties window, shown in Figure 7-1.
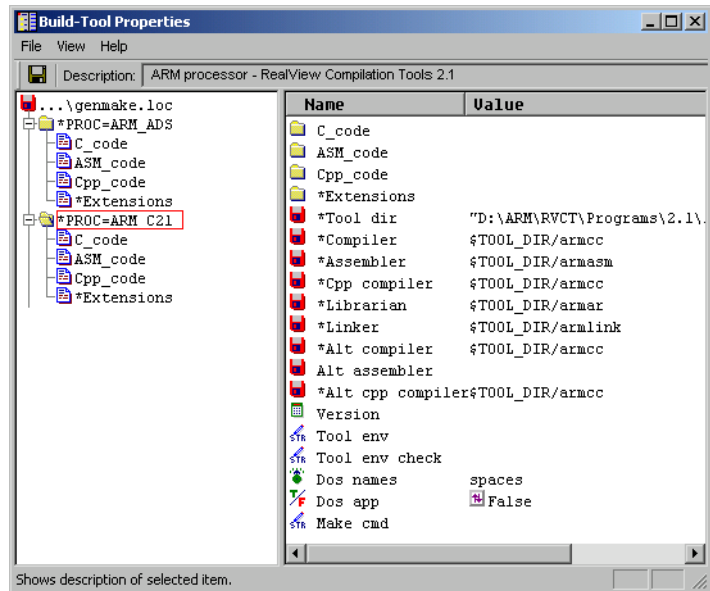
Figure 7-1 Build-Tool Properties window

The Build-Tool Properties window enables you to view the current build tools and to edit these settings to specify other tools. The entries displayed in this window depend on the type of installation you choose and the licenses you have. You can also set up your own entries to customize your projects to use specific toolsets (see *Defining build tools* on page 7-7 for details).

7.3.1   Genmake location file

The *genmake location file*, genmake.loc, makes the link between the project properties and the toolchain settings to use. For example, if you specify a particular compiler option, this file ensures that the correct option is used for your toolchain to get the result that you want.

The default file is stored in \etc. When the debugger runs for the first time, RealView Debugger copies this file into your home directory. From that point on, RealView Debugger automatically looks for this file in your home directory at each start-up. The search expands to other directories if this file is missing. This file is prebuilt with the appropriate entries for using RealView Debugger with your licensed target processors.

7.4 Defining build tools

If RealView Debugger is unable to determine the path to your build tools, or you want to override this setting, you must define the tools used to build your executable files.

To specify a build tool:

1. Select **Project** → **Build-Tool Properties...** to display the Build-Tool Properties window.

2. Select the PROC= group that you want to change in the List of Entries pane, the left pane.

   For example, click on PROC=ARM_C22 if you have RVDK. This node is expanded so that the contents are visible in the Settings Values pane, shown in Figure 7-1 on page 7-6.

3. Right-click on the setting you want to change, for example Compiler, in the Settings Values pane, the right pane.

4. Select **Edit as Filename** from the context menu to display the Enter New Filename dialog box. Locate the compiler that you want to use.

   If you know the full pathname, select **Edit Value** from the context menu, and enter the required pathname.

   Click **Save** to confirm the setting.

5. Select **File** → **Save and Close** to close the Build-Tool Properties window and update the genmake.loc file.

7.4.1 Working with open projects

If you change build tools with user-defined projects open, closing the Build-Tool Properties window displays a selection box where you can specify which projects are updated.

Select the projects to which RealView Debugger must apply the new setting and click **OK**. Click **Cancel** to leave the projects unchanged. RealView Debugger regenerates the makefile(s) for the chosen project(s). Wait for this to complete before performing any more project operations.

——— **Note** ———

The selection box shows all open projects and, therefore, might contain projects where the new build tool is not applicable.

7.5   Building files and images

For a project, you can build an executable image or a single object file using the **Tools** menu. If you have a Library project, you can build the library file or an individual object file.

——— **Note** ———

If you do not have a project open, you can still use the build options on the **Tools** menu.

This section describes:

- *Building an object file*
- *Building an executable image or library file* on page 7-9
- *Resetting line numbers* on page 7-10.

7.5.1   Building an object file

To build an object file:

1.   Open a source file so that it is displayed in the File Editor pane. This can be a C, C++, or assembly language file.

2.   Select **Tools** → **Build This File** to display the Build dialog box shown in Figure 7-2.
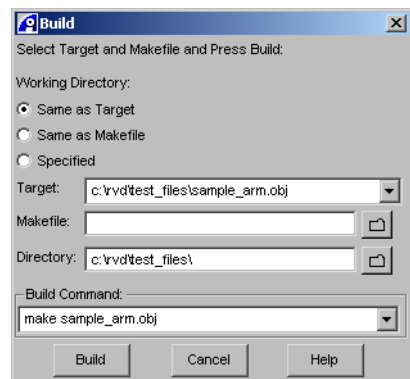
Figure 7-2 Building an object file

Entries might be preloaded into fields in the dialog box when it first opens. In this example, the file is associated with a user-defined project.

3.   Enter the build details (see *Using the Build dialog box* on page 7-11).

4.   Click **Build** to build the object file.

Alternatively, click **Cancel** to close the dialog box without starting the build.

5. After the object file is built, you can include it into an executable file for later loading to your debug target.

7.5.2 Building an executable image or library file

To build an image:

1. Ensure that there is no open project and no files in the File Editor pane.

2. Select **Tools** → **Build...** to display the Build dialog box shown in Figure 7-3.
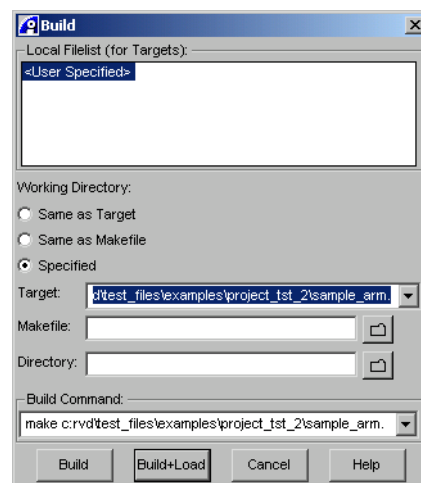
Figure 7-3 Building an image

Entries might be preloaded into fields in the dialog box when it first opens.

3. Enter the build details.

4. Click either **Build** or **Build+Load**.

Alternatively, click **Cancel** to close the dialog box without starting the build.

See *Using the Build dialog box* on page 7-11 for more details on using this dialog box.

7.5.3   Resetting line numbers

You can continue to edit a source file, that is to change, add, or delete lines of code while a project is being built. However, RealView Debugger reports any build errors or warnings using the line numbers and so this might mean that line numbers do not correspond (see *Finding build errors* on page 7-13 for details).

You can define how the debugger handles line numbers when building starts:

* RealView Debugger can use the original line numbers, that is it displays the line numbers as they were before you started to edit the source file. This means that any error messages, or build warnings, refer to the original line numbers and so debugging can continue while you are editing but makes it easier to track build errors or warnings.

* RealView Debugger can reset the line numbers so that it uses the line numbers as they currently stand, that is they reflect any changes you made since building started. This is the default behavior.

To change the way RealView Debugger handles line numbers, you must set up your **Editing Controls**:

1.   Select **Edit → Editing Controls** to enable the **Editing Controls** menu.

2.   Select **Show Original Line Numbers** so that line numbers are not reset during the build.

If you do not change this behavior, building automatically resets the original line numbers of the source code for all files in the Code window. If other files are showing in other Code windows that are part of the build, their line numbers are not reset.

———— **Note** ————

Ensure that you have a source file displayed in the File Editor pane to enable the **Editing Controls** menu.

7.6 Using the Build dialog box

The Build dialog box enables you to build images, object files, or library files from the default Code window. RealView Debugger preloads fields in the dialog box if it can. The controls and options available depend on:

- the type of target you are building
- which option you select from the **Tools** menu
- what files are displayed in the File Editor pane
- the project environment, that is what projects are open.

The Build dialog box controls are:

Local Filelist (for Targets)

Lists the currently loaded images, if any. This enables you to make a quick selection. Available only if you select **Tools** → **Build...** to build an executable image.

Working Directory

Use the radio buttons to specify where to set the working directory.

Target   Enter the target file, for example `sample_arm.obj`, if not preloaded from the project.

You can also click on the drop-down arrow to display a list of previously used build targets.

Makefile   Enter a makefile name, for example `sample_arm_Debug.mk`

Click on the directory icon to locate the required pathname or select from a list of previously used makefiles.

Directory   If the **Specified** radio button is selected, use this data field to enter the name of the working directory.

Click on the directory icon to locate the required directory or select from a list of previously used files.

Build Command

As you enter the build details, the command is constructed and displayed in this data field. You can amend the command line before submission.

You can also click on the drop-down arrow to display a list of previously used command lines.

Build   Builds the object file or executable image. Depending on the target, this also completes the compile and assemble stages.

Copyright © 2005, 2006 ARM Limited. All rights reserved.

Build+Load Builds the executable image. If the image is built successfully, RealView Debugger loads the image to the debug target processor. Available only if you select **Tools** → **Build...** to build an executable image.

Cancel Closes the dialog box without starting the build.

 ARM DUI 0291C

7.7 Finding build errors

The build error reporting system in RealView Debugger:

1. Lists the errors in the **Build** tab of the Output pane.

2. Opens the source file containing the error automatically, ready for you to correct the error.

   This only happens if the source file containing the error is not already displayed in the File Editor pane.

3. Positions the flashing text cursor in the source file tab of the File Editor pane, at or near the line containing the error. A blue arrow in the left margin of the File Editor pane also indicates this position.

When you have corrected the first error, and RealView Debugger has found more than one error, locate the next error in the source file using one of the following methods:

- select **Tools → Next Line/Error** to move through the error list
- double-click on the line number shown in the Output pane **Build** tab.

7.8   Using a build model

After you have created your user-defined project, you have to build an executable file or image from the project components. When you create a Standard project, all these stages are completed automatically from RealView Debugger presets. You can, however, manually create a project when you want to have more control over the build process.

Also, each time you make a change to settings using the Project Properties window, the makefiles are updated. The next time you build the image, RealView Debugger asks for confirmation before rebuilding.

———— **Note** ————

You do not have to have a project open to build an image. RealView Debugger provides a Build dialog box where you can specify what to build (see *Building files and images* on page 7-8 for details).

For a Standard or Library project, the build model specifies how the image is built, that is the *build target configuration*. The Project Properties window enables you to view, and change, the rules governing the build model as defined in the makefile generated by RealView Debugger.

7.8.1   Defining rules for a build model

You define the rules for a build model using groups of settings in the Project Properties window:

CONFIGURATION

Define build target configurations in this group.

COMPILE   Specify compiler options in this group.

ASSEMBLE

Specify assembler options in this group.

CUSTOM   Specify custom build options in this group.

BUILD   For a Standard project, specify linker options in this group.

BUILD_LIB For a Library project, specify library options in this group.

In addition, you can use your own makefiles for the build or use your own build tool automatically.

For more details on project properties and build target configurations see:

- Chapter 4 *Using the Project Properties Window*
- Chapter 5 *Customizing Projects*
- Chapter 8 *Managing Build Target Configurations*.

ARM DUI 0291C

Chapter 8
Managing Build Target Configurations

This chapter describes how to manage build target configurations for user-defined projects in RealView® Debugger. It contains the following sections:

- *About build target configurations* on page 8-2
- *Working with the examples* on page 8-5
- *Working with build target configurations* on page 8-6
- *Working with settings in build target configurations* on page 8-9.

8.1 About build target configurations

A user-defined project defines at least one build target configuration, for example a `Debug` or a `Release` build. For ARM® architecture-based projects, RealView Debugger defines three build target configurations:

- Debug
- DebugRel
- Release.

See *Build target configurations* on page 1-11 for a description of these configurations.

Build target configurations can share files in the same project, while using their own build settings. You can also define a specific build order for the build target configurations in a project. The Project Properties window in RealView Debugger enables you to define and set up such relationships.

Each build target configuration has a corresponding directory where the built files are placed. The directories have the same name as the build target configuration, and are subdirectories of your top-level project base directory.

This section describes:

- *Defining build target configurations*
- *Base settings*
- *Viewing build target configurations* on page 8-3
- *Active build target configuration* on page 8-4.

8.1.1 Defining build target configurations

For each project, build target configurations are defined in the `CONFIGURATION` group and saved in the project settings file. These special child groups are then replicated through the settings hierarchy to other groups in the project. Where a group contains child groups, for example the `COMPILE=arm` group (see Figure 8-1 on page 8-3), these build target configurations entries are also created in the parent group and all child groups.

If you define a new build target configuration this is immediately replicated to these other groups. It is not necessary to save the project to make this happen. See *Creating a configuration* on page 8-6 for an example of how to do this.

8.1.2 Base settings

For every project you create, each parent group also contains *base settings* that apply to one or all of the build target configurations, shown in Figure 8-1 on page 8-3. You can customize your project so that settings apply to all the build target configuration groups or only to specific groups within the parent group.

8.1.3   Viewing build target configurations

Figure 8-1 shows the *COMPILE=arm group for the example dhrystone project. This group contains settings and groups of settings that define the build model for the ARM C compiler.
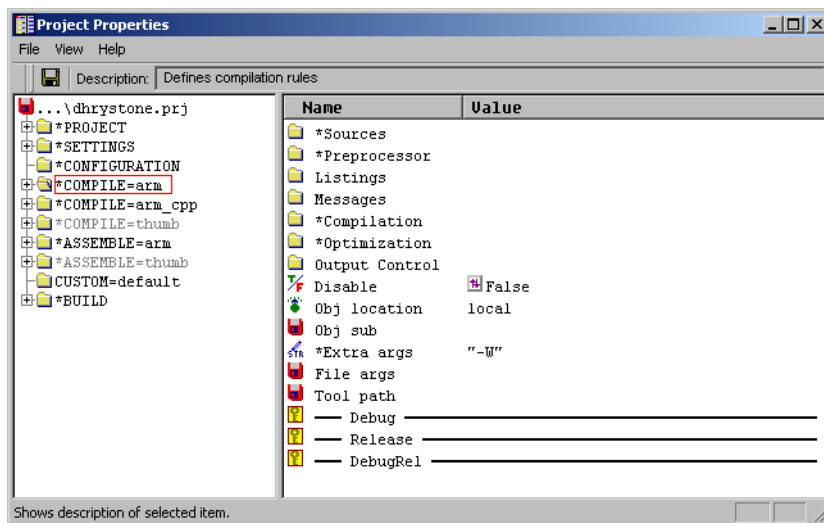
Figure 8-1 Base settings and build target configurations for a COMPILE group

The right pane shows the contents of this group:

*   Base settings used in one or more build target configurations. These are single settings, for example Obj_location, or groups of settings, for example Compilation.

    If you specify a base setting, this is applied across all the build target configurations in the group. However, you can define specific settings for individual configurations within the group (see *Assigning a specific setting to a configuration* on page 8-9 for details).

*   Build target configuration groups used to hold configuration-specific settings, for example the DebugRel group.

    A special icon identifies these groups because they are internal groups generated by RealView Debugger.

8.1.4   Active build target configuration

If you have more than one build target configuration, you must specify the configuration that RealView Debugger uses when building your application. This is known as the *active build target configuration*. See *Changing the active configuration* on page 8-7 for details on how to change the active build target configuration.

                   ARM DUI 0291C

8.2 Working with the examples

This chapter contains examples of making changes to the user-defined project dhrystone.prj installed in the *install_directory*\RVDK\Examples\... directory. You might want to make a backup of the project base directory before following the examples so that the default files and settings can be restored.

——— **Caution** ———

The changes described here conflict with the default build target configurations as specified in the example project. These examples are included only to show how to amend project and build-tool properties.

It is recommended that you read this section before following the examples:
- *Before you start*
- *Connecting to a debug target*
- *Changing settings*.

8.2.1 Before you start

To start working with the example dhrystone project:

1. Make a copy of the project base directory if necessary.

2. Start RealView Debugger.

3. Select **Project** → **Open Project...** from the default Code window main menu.

4. Locate the project settings file dhrystone.prj and click **Open**.

8.2.2 Connecting to a debug target

These examples assume that you are not connected to a debug target. This means that RealView Debugger does not try to set default binding (see Chapter 9 *Project Binding* for more details).

8.2.3 Changing settings

These examples describe how to change project settings, using the Project Properties window. For details on the layout of this window see *Project Properties window* on page 4-3. These examples assume you are familiar with the procedures described in the online help topic *Changing Settings*.

8.3   Working with build target configurations

The CONFIGURATION group, shown in Figure 8-1 on page 8-3, defines the build target configurations for the project. The examples in this section describe operations on settings in this group:

- *Creating a configuration*
- *Deleting a configuration* on page 8-7
- *Changing the active configuration* on page 8-7
- *Changing the order of configurations* on page 8-8.

8.3.1   Creating a configuration

To create a new build target configuration:

1.   Select **Project** → **Project Properties...** to display the Project Properties window.

2.   Select the ∗CONFIGURATION group in the List of Entries pane, the left pane.

3.   Right-click on the Config setting at the top of the Settings Values pane, the right pane, and select **Make New...** from the context menu.

4.   Type the name of your new configuration, for example myBuilds, then press Enter.

A new ∗Config setting is created in the CONFIGURATION group.

This adds a new group, called myBuilds, to every build target configuration group specified for the project. This applies to disabled groups.

To see the change, right-click on a ∗COMPILE= group, in the left pane, and select **Explore** from the context menu. The new myBuilds entry is visible in the build target configurations groups in the right pane.

5.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

RealView Debugger creates a makefile (for example, dhrystone_myBuilds.mk) and a subdirectory for the new build target configuration in the project base directory.

6.   Select **Tools** → **Build...** to rebuild the application.

                                       ARM DUI 0291C

8.3.2   Deleting a configuration

To delete a build target configuration:

1.   Select **Project** → **Project Properties...** to display the Project Properties window.

2.   Select the *CONFIGURATION group in the List of Entries pane.

3.   Right-click on the *Config setting to be removed and select **Delete**.

      This deletes the specified group from the CONFIGURATION group. The group is also removed from every build target configuration group for the project.

      ——— **Note** ———
      The makefile and build directory for the deleted configuration are not deleted, where they exist.
      —————————————

4.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

5.   Select **Tools** → **Build...** to rebuild the application.

8.3.3   Changing the active configuration

By default, the active build target configuration is defined by the top-most entry in the list of *Config settings in the CONFIGURATION group. In the dhrystone project this is Debug If you change the order of configurations (see *Changing the order of configurations* on page 8-8) then the active build target configuration also changes.

You can specify which build target configuration is the active configuration and so override the default:

1.   Select **Project** → **Project Properties...** to display the Project Properties window.

2.   Select the *CONFIGURATION group in the List of Entries pane.

3.   Right-click on the Active_config setting in the Settings Values pane, and select the required configuration from the options list, for example **DebugRel**.

4.   Select **File** → **Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

5.   Select **Tools** → **Build...** to rebuild the application.

8.3.4 Changing the order of configurations

If you work with build target configurations in a particular sequence, you can arrange the settings so they appear in that sequence in the Project Properties window. For example, you might start by building a Debug configuration, then a DebugRel configuration, and finally a Release version.

To change the order of the build target configurations:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Select the *CONFIGURATION group in the List of Entries pane.

3. Right-click on any *Config setting in the Settings Values pane, and select **Manage List...** from the context menu.

    The Settings: List Manager dialog box is displayed.

4. Select a build target configuration that you want to move, so that the associated tick box is checked.

5. Click **Move Up** or **Move Down** to move the select configuration as required.

    Repeat these steps for each configuration you want to move.

6. Click **OK** to confirm the order and close the Settings: List Manager dialog box.

7. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

8. Select **Tools → Build...** to rebuild the application.

This does not change the order of build target configuration entries in the project groups.

8.4 Working with settings in build target configurations

The examples in this section describe operations on settings assigned to build target configurations:

- *Assigning a specific setting to a configuration*
- *Removing a setting from a configuration* on page 8-10
- *Assigning multiple settings to a specific configuration* on page 8-12.

8.4.1 Assigning a specific setting to a configuration

Each build target configuration can be assigned a different value for a particular setting. For example, you might want to have a different value for the Speed_vs_space setting in each configuration.

——— **Note** ———

For projects that support build target configurations, the context menu for settings in the Settings Values pane includes the option **Move/Copy to Configuration...**. This option is valid only for the settings in the COMPILE, ASSEMBLE, CUSTOM, BUILD and BUILD_LIB groups. You must not use this option on settings in the PROJECT, SETTINGS, or CONFIGURATION groups.

To assign a specific setting value to a build target configuration:

1. Select **Project → Project Properties...** to display the Project Properties window.

2. Select the group containing the setting to be assigned. For example:

   a. Select the *COMPILE=arm group in the List of Entries pane.

   b. Right-click on the *Optimization group in the Settings Values pane, and select **Explore** from the context menu.

3. Modify the setting to be assigned. For example:

   a. Right-click on the Speed_vs_space setting in the Settings Values pane.

   b. Select **space** from the context menu.

   ——— **Note** ———

   If you want to copy the setting to more than one build target configuration, you must modify the setting first. Only settings that have been changed from the default can be copied to other configurations.

4. Right-click on the Speed_vs_space setting, and select **Move/Copy to Configuration...** from the context menu.

A list selection box is displayed that shows the actions available. There is a move and copy action for each of your build target configurations.

5.   Select the required action, for example **Copy to Debug**.

   •   If you have not modified the setting from the original default value, then the setting is always *moved* to the build target configuration, even if you choose to copy the setting.

   •   If you have modified the setting, then the action you selected is performed.

   In this example, the setting has been modified from the default and so a copy of the setting is assigned to the Debug build target configuration.

6.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7.   Select **Tools → Build...** to rebuild the application.

——— **Note** ———

This change only applies to this *Optimization group. It does not replicate throughout other groups in the project.

8.4.2   Removing a setting from a configuration

When you assign a setting to a build target configuration, and an instance of the setting also exists in the base settings, the assigned setting value overrides the base settings value. If you want the base settings value to be used for the configuration, you can remove the assigned setting.

For example, if you have completed the procedure described in *Assigning a specific setting to a configuration* on page 8-9, you might now want to remove the Speed_vs_space setting.

To remove a setting from a build target configuration:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the group containing the setting to be removed. For example:

   a.   Select the *COMPILE=arm group in the List of Entries pane.

   b.   Right-click on the *Optimization group in the Settings Values pane, and select **Explore** from the context menu.

3.   Right-click on the Speed_vs_space setting in the Debug build target configuration to display the context menu.

4.   Select an option to reset the setting:

- •   Select **Move/Copy to Configuration...** if you have not modified this instance of the setting from the default value. This enables you to move the setting into the base settings (see *Moving a setting into the base settings*).

- •   Select either **Reset to Default** or **Reset to Empty** if you have modified this instance of the setting from the default value. Only one of these options is available.

  The result of selecting a **Reset to ...** option depends on other settings in this group:

  — If there are other instances of the chosen setting in this settings group, this instance of the setting is deleted from the build target configuration.

  — If this is the only instance of the chosen setting in this settings group, the value of the setting is changed according to the menu option selected. This instance of the setting remains in the build target configuration. That is, there is no instance in the base settings. Therefore, you might want to move the setting back into the base settings (see *Moving a setting into the base settings*).

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

6.   Select **Tools → Build...** to rebuild the application.

Moving a setting into the base settings

If there is no instance of a setting in the base settings for a settings group, and you want that setting to apply to more than one build target configuration, then you must move the setting into the base settings. For example you might want to move a setting from the DebugRel configuration so that it applies to all build target configurations in the parent group.

To move a setting from a build target configuration to the base settings:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the group containing the setting to be moved. For example:

   a.   Select the *COMIPILE=arm group in the List of Entries pane.

   b.   Right-click on the *Optimization group in the Settings Values pane, and select **Explore** from the context menu.

3.   Right-click on the setting in the required build target configuration. For example:

    a.   Right-click on the *Debug_optimize partial setting in the DebugRel
          configuration.

    b.   Select **Move/Copy to Configuration...** from the context menu.

4.   Select Move to <Base> from the selection box, and click **OK**.

5.   Select **File → Save and Close** to regenerate the makefile(s) for the project, and
      close the Project Properties window.

6.   Select **Tools → Build...** to rebuild the application.

8.4.3   Assigning multiple settings to a specific configuration

To assign multiple settings to a build target configuration:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the group containing the setting to be assigned. For example:

    a.   Select the *COMPILE=arm group in the List of Entries pane.

    b.   Right-click on the *Optimization group in the Settings Values pane, and
          select **Explore** from the context menu.

3.   Right-click on the build target configuration group to which the settings are to be
      assigned, for example Debug, and select the action you require from the context
      menu:

    •   **Insert Item(s) into this Configuration...**
         Inserts a new instance of a setting into the specified configuration. The
         value of the new instance is the default value of the setting. The original
         instance of the setting remains unchanged.

    •   **Move Item(s) into this Configuration...**
         Moves a current instance of a setting into the specified configuration. This
         maintains the current value of the setting.

    •   **Copy Item(s) into this Configuration...**
         Copies a current instance of a setting into the specified configuration. This
         maintains the current value of the setting.

4.   Select the setting(s) to be assigned to the chosen build target configuration so that
      the associated tick boxes are checked:

    •   if you choose to insert settings, the default settings are listed

    •   if you choose to move or copy settings, then only the modified settings are
         listed, if any.

———— **Note** ————

If there is more than one instance of a setting, then each instance of the setting is listed. Settings are applied in the same order that they appear in the Settings Values pane, that is the first setting in the list takes precedence. Unused settings are marked with an exclamation mark and are grayed out.

5. Click **OK** to confirm your choices and assign the setting(s).

6. Select **File → Save and Close** to regenerate the makefile(s) for the project, and close the Project Properties window.

7. Select **Tools → Build...** to rebuild the application.

Chapter 9
Project Binding

This chapter describes project binding in RealView® Debugger in more detail. It contains the following sections:

- *Types of binding* on page 9-2
- *Viewing project binding* on page 9-7
- *Forcing binding* on page 9-9
- *Effects of binding* on page 9-12
- *Effects of unbinding* on page 9-13
- *Connecting and disconnecting* on page 9-14.

9.1 Types of binding

RealView Debugger *binds* projects to connections by associating the image for a user-defined project or auto-project with an available debug target that has a corresponding processor type. The binding mechanism can also load the project image automatically, and execute RealView Debugger commands, depending on settings in the project settings file.

RealView Debugger binds a project using either:

* *Default binding*
* *Autobinding* on page 9-3.

9.1.1 Default binding

When you first create a Standard, Library, or Custom project, you define the toolchain, associated with the target application using the Toolchain drop-down list box on the Create New Project dialog box. This entry, in the project settings file, determines the default binding for the project.

――――― **Note** ―――――

RealView Debugger populates this setting automatically when it creates an in-memory auto-project for an image that you load directly to a debug target.

If you connect to a debug target and open a user-defined project, RealView Debugger attempts to bind the project by default:

* If the connection does not correspond to the processor family for the project, the project opens but it does not bind. If the project does not bind, the image name is not registered. This means that the debugger cannot autoload the image on a GO or RELOAD command.

* If the connection corresponds to the processor family and there is no project already bound to the connection, RealView Debugger binds the project by default.

* If the connection corresponds to the processor family and there is a project already bound to the connection by default, RealView Debugger gives you the option to unbind the active project and bind the new project.

  If you choose not to bind the new project, the project opens but is unbound.

* If the connection corresponds to the processor family and there is a project already autobound to the connection, RealView Debugger displays a warning that it cannot complete the binding. This is because default binding cannot displace an autobound project. The project opens but is unbound. See *Autobinding* on page 9-3 for details about autobound projects.

This behavior means that you can always open a user-defined project, and make changes to the project settings file, even if your connection is not a suitable debug target for that project.

See *Effects of binding* on page 9-12 and *Effects of unbinding* on page 9-13 for more details on what happens when binding changes.

9.1.2   Autobinding

You can specify that a project can only bind to a specific processor or device, by name, using the Specific_device entry in the project settings file, shown in Figure 9-1. You can do this when you first create the project or update this setting later.
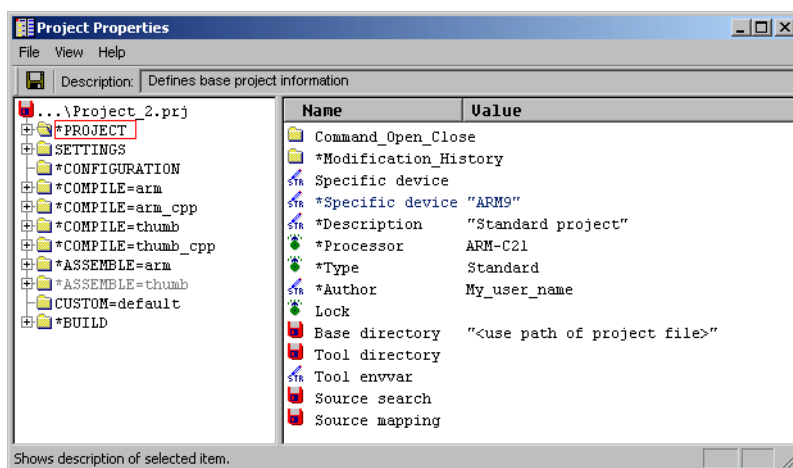
Figure 9-1 Specifying autobinding

This close coupling of project and target processor is called *specific device binding* and forces RealView Debugger to use *autobinding* when the project opens. If autobound, the project only binds to the named device. If this is not set, RealView Debugger uses default binding to determine project binding.

——— **Note** ———

You cannot force an autobound project to bind to a device that does not match. You must change the Specific_device setting first, see *Adding Specific_device settings* on page 9-4 for details. It is not necessary to reopen the project before rebinding. See *Forcing binding* on page 9-9 for details on defining binding behavior.

———

Autobinding gives a project priority over other projects when it opens to a connection:

• If the connection does not correspond to the named device, the project opens but it does not bind.

• If the connection corresponds to the named device and there is no project already bound to the connection, RealView Debugger autobinds the project.

• If the connection corresponds to the named device and there is a project already bound to the connection, using autobinding or default binding, RealView Debugger unbinds the active project and autobinds the new project. There is no warning.

• If the connection partially corresponds to the named device, for example the `Specific_device` entry is set to `ARM`, and there is a project already autobound to the connection, for example where the `Specific_device` entry is set to `ARM7`, RealView Debugger unbinds the active project and autobinds the new project. There is no warning.

If you open a user-defined project that has no `Specific_device` entry specified and there is a project already autobound to the connection, RealView Debugger displays a warning that it cannot complete the binding. This is because default binding cannot displace an autobound project.

――――― **Note** ―――――

Autobound projects might not get priority:

• if they are subprojects in a Container project, see *Working with Container projects* on page 10-7 for details

• if they are opened by the workspace, see the chapter describing workspaces in *RealView Developer Kit v2.2 Debugger User Guide* for details.

―――――――――――――

Adding Specific_device settings

The `Specific_device` setting defines an ordered device list to which a project can autobind. For example, an ARM9 core might be the first choice but an ARM7 core is the second choice.

You can also use partial matching to control how projects autobind to different processors. For example, your debug target might contain an ARM966E-S™ and an ARM926EJ-S™ core. If you specify `ARM966` the project only binds to the ARM966E-S core. However, if you specify `ARM9` the project can bind to either the ARM966E-S or the

ARM926EJ-S connection. In addition, if you have specified multiple devices where ARM9 appears before ARM966E-S, RealView Debugger attempts to bind the project using the ARM9 device first. Therefore, the project always binds to an ARM9 core.

This matching behavior means that, where you have specific device names and a partial device name, it might be useful to place the partial device name at the end of the list, as a catch-all binding. See *Changing the order of Specific_device settings* on page 9-6 for details on how to change the order of devices.

To specify autobinding:

1.   Select **Project → Project Properties...** to display the Project Properties window.

2.   Select the *PROJECT group in the List of Entries pane.

3.   Right-click on the Specific_device setting in the Settings Values pane and select **Edit Value** from the context menu.

4.   Enter the specific device name or family name (for example ARM946E-S or ARM7), then press Enter to complete the entry.

      A new setting is created, for example *Specific_device_ARM946E-S. Each device you add creates another entry.

5.   Select **File → Save Changes** to save your changes. The makefiles for the project are regenerated.

6.   Select **File → Close Window**.

If the project is already bound then changing the Specific_device settings value does not reset the binding. RealView Debugger only attempts to autobind when you open a project. Therefore, after changing the Specific_device settings value either:

•   reopen the project to see the change

•   use the Project Control dialog box to rebind the project (see *Forcing binding* on page 9-9 for details).

———— **Note** ————

To restore the setting, right-click on the Specific_device setting in the Settings Values pane and select **Delete** from the context menu.

Changing the order of Specific_device settings

If you have multiple Specific_device settings values, RealView Debugger attempts to autobind the project in the order they appear in the settings list. You can change the order using the Settings: List Manager dialog box:

1.   Select **Project** → **Project Properties...**to display the Project Properties window.

2.   Select the *PROJECT group in the List of Entries pane.

3.   Right-click on the Specific_device setting in the Settings Values pane and select **Manage List...** from the context menu to display the Settings: List Manager dialog box.

4.   Select the device that you want to move.

5.   Click the **Move Up** or **Move Down** button to move the device to the required position in the list.

6.   Click **OK** to close the Settings: List Manager dialog box.

7.   Select **File** → **Save Changes** to save your changes. The makefiles for the project are regenerated.

8.   Select **File** → **Close Window**.

――― **Note** ―――

When setting a Specific_device value, names must match device names stored in the corresponding JTAG configuration file. See the chapter describing configuring custom targets in *RealView Developer Kit v2.2 Debugger User Guide*.

9.2 Viewing project binding

If you create or open a user-defined project without previously connecting to a debug target, the project is unbound. This also applies if you create or open a project where the specified processor family for that project does not correspond to the current connection.

For example, if you connect to a DSP debug target, and you then create a user-defined project for an ARM processor, this new project is shown as unbound in the default Code window title bar:

```
RVDEBUG<ARM-Project_1> = @DSPconnection [Unattached]
```

——— **Note** ———
RVDK does not support DSPs.

If you are connected to a target when you create or open a user-defined project, RealView Debugger binds the project to the connection if it can. When you load an image to create an in-memory auto-project, or if RealView Debugger opens the saved auto-project associated with the image, the auto-project binds automatically.

The Code window title bar shows that the project is bound to a connection by including the project name in parentheses, for example:

```
RVDEBUG(ARM-Project_2) = @ARM946E-S_0:ARM-ARM-USB [Unattached]
```

——— **Note** ———
For an auto-project, the name in the title bar is not the image name, but the project name created from the image name.

The title bar updates if you change the project environment, for example if you:
- use the Project Control dialog box to unbind or rebind a project
- use the Project Control dialog box to change the active project
- open and bind a new project
- disconnect.

Use the Project Control dialog box to see the open project list and how projects are bound. See the example in Figure 9-2 on page 9-8.

9.2.1 Viewing autobound projects

The Code window title bar shows if a project is bound to the current connection. The title bar does not show if the project is autobound. If you do not want to examine the project properties, you can use the Project Control dialog box to see if a project has specific device binding set.

Any open projects that are autobound have (DEV) appended to the entry in the open project list, shown in Figure 9-2.
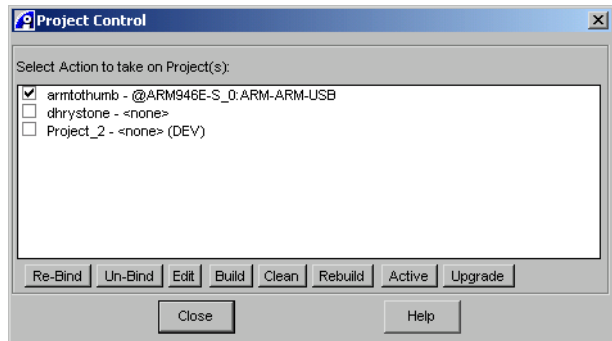
Figure 9-2 Autobound projects in the Project Control

In Figure 9-2, there are three projects in the open project list:

- The last project to open is at the top of the list and is selected by default (armtothumb). This is bound to the current connection, using default binding.

- The dhrystone project is open but is not bound to a connection.

- The Project_2 project is autobound but is not bound because there is no matching connection. You can see that the project is autobound because (DEV) is appended to the entry to show that the project has specific device binding set.

In this case, armtothumb is the active project because it was the last project to bind successfully to the current connection.

9.3 Forcing binding

Select **Project** → **Project Control...** from the default Code window main menu to display the Project Control dialog box where you can force project binding and so change your project environment. When you are working with multiple projects, you can use the Project Control dialog box to control project binding so that you have full control over your project environment.

——— **Note** ———

See *Working with the Project Control dialog box* on page 6-3 for details on other controls in this dialog box.

In Figure 9-2 on page 9-8, there are three projects in the open project list. The last project to open is shown at the top of the list and is selected by default (armtothumb). This project is bound to the current connection, using default binding. The dhrystone project is open but unbound.

In this example, armtothumb is the active project because it was the last project to bind successfully to the current connection. This means that you can access the project properties from the **Project** menu or from the Process Control pane, if this is visible.

To bind the dhrystone project to the current connection:

1.   Select the armtothumb project so that it is unticked.

2.   Select the dhrystone project so that it is ticked. This selects the project for action.

3.   Click **Re-Bind** to force RealView Debugger to bind the selected project. This displays a list selection box showing the current connection, shown in Figure 9-3.
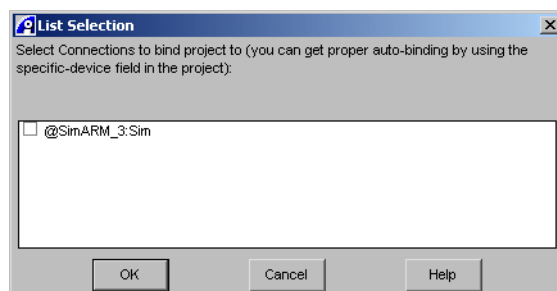
Figure 9-3 Connection selection box

4.   Select the connection so that it is ticked. You must do this even where the list contains only one connection.

---

5. Click **OK** to confirm your choice and close the selection box. Otherwise, click **Cancel** if you want to abort the binding operation.

   This updates the contents of the Project Control dialog box to show the new binding.

6. Click **Close** to close the Project Control dialog box.

See *Effects of binding* on page 9-12 for what happens when a project binds.

9.3.1 Autobound projects

You can use the Project Control dialog box to force binding when you are working with autobound projects. If, for example, the open project list contains four projects where three have a `Specific_device` entry set and one uses default binding, you can select all four projects and then click **Re-Bind** to force binding. Each project is then bound in turn. You can, therefore, use this dialog box to force default binding to displace an autobound project.

―――― **Note** ――――

You cannot use the Project Control dialog box to force an autobound project to bind to a device that does not match. You must change the `Specific_device` setting first, see *Adding Specific_device settings* on page 9-4 for details.

9.3.2 Forcing unbinding

You do not have to unbind a project before you rebind another project to the same connection. However, you might want to unbind a project so that you can open a new project and ensure that it binds to the connection, for example if you have an autobound project bound to the current connection and you want to open a project that uses default binding. Use the Project Control dialog box to unbind projects. Do this in the same way as rebinding but click **Un-Bind** to unbind the selected project(s). See *Effects of unbinding* on page 9-13 for what happens when a project unbinds.

If you unbind a user-defined project where the image associated with that project is loaded to the debug target, RealView Debugger tries to use an auto-project to store settings for the image. If you then open and bind another project to this connection, RealView Debugger displays both projects in the Process Control pane. In this case, the Project Control dialog box shows three projects:

- the auto-project associated with the current image
- the second user-defined project bound to the current connection
- the user-defined project opened first.

See Chapter 10 *Working with Multiple Projects* for details on working with multiple projects in this way.

9.4 Effects of binding

When a project binds to a connection:

- Any load-related commands specified in the SETTINGS group are executed, for example loading the image, setting top of memory, or semihosting.

  As a minimum, binding registers the image details with RealView Debugger. This means that the debugger can autoload the image on a GO or RELOAD command.

- Any open commands you have specified are executed. These are defined in the project settings file in the Command_Open_Close group in the top-level PROJECT group.

  RealView Debugger also executes these commands if you use the Project Control dialog box to rebind the project (see *Forcing binding* on page 9-9).

- RealView Debugger updates the default Code window title bar.

- If the Process Control pane is visible, the display in the **Process** tab is updated with project details.

- If you force a project to bind to a connection and there is already a project bound to that connection, the first project unbinds. This applies even where the first project is autobound. See *Effects of unbinding* on page 9-13 for details on what happens when the first project unbinds.

               ARM DUI 0291C

9.5 Effects of unbinding

When a project unbinds from a connection:

- RealView Debugger updates the default Code window title bar.

- If the Process Control pane is visible, all process details are removed from the **Process** tab.

- Any close commands you have specified are executed. These are defined in the project settings file in the Command_Open_Close group in the top-level PROJECT group.

  RealView Debugger also executes these commands if you use the Project Control dialog box to unbind the project (see *Forcing unbinding* on page 9-10).

9.6   Connecting and disconnecting

Connecting to, and disconnecting from, a debug target changes the project environment:

- If you open a project and then connect to a target, RealView Debugger tries to bind the project to the connection in the normal way.

- If you open multiple projects and then connect to a target, RealView Debugger binds any autobound projects first. Default binding is used where there are no autobound projects.

- If you connect to a target, this might change the active project depending on the current project environment.

- If you connect to a target, RealView Debugger updates the default Code window title bar.

- If you connect to a target and a project binds, RealView Debugger updates the Process Control pane, if this is visible.

- If you connect to a target and a project binds, any open commands you have specified as part of the project definition are executed.

- If you disconnect, this might change the active project depending on the current project environment.

- If you disconnect, RealView Debugger updates the default Code window title bar.

- If you disconnect, RealView Debugger clears the Process Control pane.

- If you disconnect and there is a project bound to the connection, then the project unbinds but any close commands are not run because the connection has been lost.

- If you disconnect, this does not close any open projects. This means that you can continue to make changes to the project properties. This applies to user-defined projects and auto-projects.

See Chapter 10 *Working with Multiple Projects* for details on what else happens if you are working with multiple projects and you connect or disconnect.

             ARM DUI 0291C

Chapter 10
Working with Multiple Projects

This chapter describes how to work with multiple projects in a RealView® Debugger debugging session. It assumes that you are familiar with the concepts and terms described in all the previous chapters in this book.

This chapter contains the following sections:

10.1 Working with multiple projects

Where you are working with a single project, it is normally bound to the connection and so you can perform any action on it easily.

Where you are working with multiple projects, the project environment depends on:
- the order in which projects open
- project binding
- any loaded images.

This rest of this chapter describes how to work with multiple projects in a debugging session. It provides hints and tips if you are working with:
- multiple user-defined projects
- Container projects
- multiple auto-projects
- mixed user-defined projects and auto-projects.

If you are working with a single user-defined project, where the image associated with that project is loaded to the debug target, and you unbind the project, RealView Debugger tries to use an auto-project to store settings for the image. If you then rebind the user-defined project, it replaces the auto-project in the Process Control pane.

---— **Note** ———

It is recommended that you do not do this. Always use the user-defined project for an image that you load, where this exists. Only use an auto-project for an image where no user-defined project exists.

If you unbind a user-defined project where the image associated with that project is loaded to the debug target, RealView Debugger uses an auto-project to store settings for the image. If you then bind a different project to this connection, the Process Control pane shows details of both projects. See *Working with auto-projects* on page 10-10 for details.

Therefore, when you are working with several open projects, remember:

- By default, the project shown in the default Code window title bar is the active project. If you are not connected to a debug target, this defaults to the last project you opened. If you are connected, this is the last project to bind successfully. See *Active projects* on page 10-4 for more details on active projects.

- Project-level commands selected from the **Project** and **Tools** menus act on the active project only.

- The **Process** tab, in the Process Control pane, shows details about the bound project and gives you access to the project properties. This can also be used to load and unload the image associated with the project.

- If you are connected, an open project that is unbound is usually not the active project. This means that you cannot access the project properties from the **Project** menu, or from the Process Control pane.

  Use the Project Control dialog box to work on an unbound project without changing the project environment, for example where you do not want to rebind to a connection. See *Using the Project Control dialog box* on page 6-2 for details.

- The Process Control pane might show details for more than one process depending on the properties of the project or the state of your target. See *Working with auto-projects* on page 10-10 for more details.

10.2   Active projects

When working with multiple projects, the active project is the project that you can modify or build in RealView Debugger:

•   There can be only one active project for a connection.

•   The active project name appears in the default Code window title bar.

•   The active project is usually bound to the connection.

•   The project base directory for the active project usually defines the current working directory.

You can perform any action on this project. However, you might have to perform actions on other projects that you have opened. RealView Debugger enables you to change the project shown in the Code window title bar, so that you can perform actions on a different project. That is, you can make another project the active project. See *Changing the active project* for more details.

10.2.1   Changing the active project

In Figure 10-1, there are three projects in the open project list. The last project to open is shown at the top of the list and is selected by default. The primes project is bound to the current connection, using default binding. The dhrystone project is open but unbound.
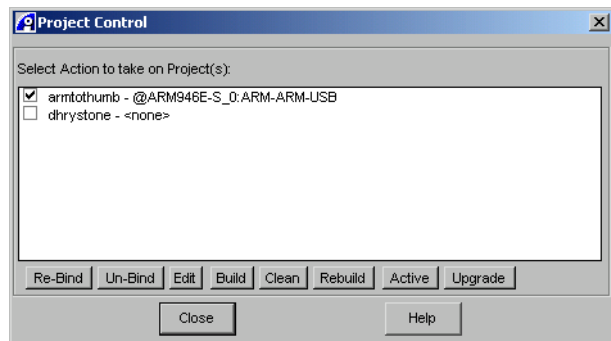
Figure 10-1 Project Control dialog box

In this case, primes is the active project because it was the last project to bind successfully to the current connection. This means that you can access the project properties from the **Project** menu or from the Process Control pane, if this is visible.

                    ARM DUI 0291C

To change the active project:

1. Close any open source files in the File Editor pane. This is to ensure that you do not have any files open that are for other projects if you use the **Project** menu to execute project-level commands, for example adding source files.

2. Select **Project → Project Control...** from the default Code window main menu, to display the Project Control dialog box.

3. Click the check box for the current active project (armtothumb) so that it is not checked.

4. Click the check box for the project that you want to make the active project (dhrystone).

5. Click **Active**.

    The project entry is moved to the beginning of the open project list. An asterisk (*) is placed before the project entry to show that this is now the active project and that it has changed from the default.

    The default Code window title bar changes to show the new active project and that the project is unbound, for example <dhrystone>.

6. Click **Close** to close the Project Control dialog box.

 ——— **Note** ———

Project binding is not affected when you change the active project in this way. This means that, in this example, the Process Control pane continues to show the project that is bound to the current connection, for example primes.

If you select multiple projects from the list and then click an action control, the specified action is carried out on the selected projects in the order they are listed. This means that, in some cases, the last action completed successfully overwrites all previous actions. For example, if you have three projects open, deselect the first project in the list, select the last two projects, and then click **Active**. This makes the last selected project the active project, and RealView Debugger moves that project to the top of the list.

Using the Active button

You can use the **Active** button to make a project active so that you can access the project properties, or perform actions on that project using the **Project** or **Tools** menu.

The action of the **Active** button depends on previous operations you have carried out on the open project list. If you click the **Active** button again for a project that you have previously set to be the active project, the asterisk is removed from the project name in the Project Control dialog box:

•   If there is a project already bound to the current connection, this bound project becomes the active project.

•   If there is no project bound to the current connection, then the project at the top of the list remains the active project.

Using the **Active** button in this way:
•   does not change the order of projects in the open project list
•   does not change project binding.

 ARM DUI 0291C

10.3   Working with Container projects

A Container project uses only a subset of the project settings. The project settings file contains a single group, the PROJECT group, that holds a list of *subprojects*. When you open a Container project, RealView Debugger opens the associated subprojects, shown in Figure 10-2 on page 10-8. This means that, by definition, you are working with multiple projects.

―――― **Note** ――――

For a Container project, the order of the subproject list defines the order in which the subprojects open and build. You must specify the subproject list, therefore, in order of dependency (see *Working with the Project Control dialog box* on page 10-8 for details).

See *Steps for creating a Container project* on page 3-12 for details on how to create a Container project.

―――――――――――

This section describes working with Container projects. It contains the following sections:
*   *Opening a Container project*
*   *Working with the Project Control dialog box* on page 10-8
*   *Nesting Container projects* on page 10-9.

10.3.1   Opening a Container project

If you open a Container project:

*   The project binds to the current connection using the normal default binding rules (see *Types of binding* on page 9-2 for details).

*   All subprojects open but do not bind, even where they are autobound. RealView Debugger warns of any subprojects it cannot open.

*   The open project list is updated to show all the open projects, including subprojects.

*   The Container project becomes the default active project for the connection.

―――― **Note** ――――

A Container project cannot be autobound but subprojects, held in the PROJECT group, can define one or more specific device binding settings. Therefore, a Container project cannot open and bind to a connection if there is an autobound project already bound to that connection.

―――――――――――

10.3.2   Working with the Project Control dialog box

You can perform actions on one or more of the subprojects that make up a Container project from the Project Control dialog box, shown in Figure 10-2. Use the Project Control dialog box to access project properties for any subproject or to control subproject binding.
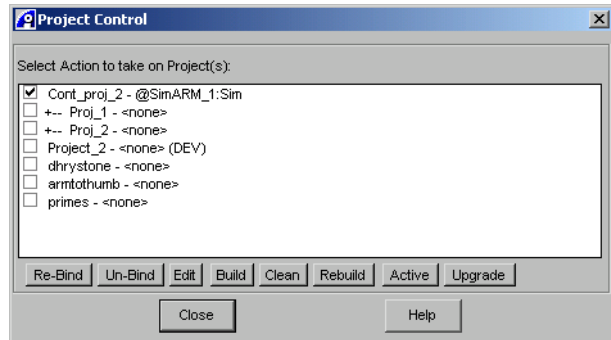
Figure 10-2 Container projects in the Project Control

In Figure 10-2, there are five projects in the open project list. The active project, the Container project `Cont_proj_2`, is shown at the top of the list and is selected by default. This project is bound to the current connection.

In addition, the Container project defines two subprojects, `Proj_1` and `Proj_2`, shown by the plus sign appended to each entry. Both are open but unbound. In this example, the two subprojects are not Container projects (see *Nesting Container projects* on page 10-9).

For a Container project, the order of the subproject list, held in the `PROJECT` group, defines the order in which the subprojects open. In this example, the subproject list contained the entries:

```
Sub_project "..\proj_2\Proj_2.prj"
Sub_project "..\proj_1\Proj_1.prj"
```

This specifies that `Proj_2` opens before `Proj_1`. However, the Project Control dialog box shows that subproject `Proj_1` was the last project to open and so this is at the top of the list of open subprojects (see Figure 10-2).

When you build your container project, the last project to open is the first project to build so the Project Control dialog box shows the build order for `Cont_proj_2` as `Proj_1` followed by `Proj_2` (see Figure 10-2). This ordering might be important where you have

dependent projects. Therefore, you must specify the subproject list in order of dependency, that is dependent projects must appear first in the list so that they are opened before, but are built after, the projects on which they depend.

——— **Note** ———

You can use the Project Control dialog box to select subprojects for building or rebuilding as required.

You cannot add files to a Container project, only to a subproject. To do this, use the Project Control to make the subproject active. For details see:

* *Changing the active project* on page 10-4
* *Adding files to a user-defined project* on page 2-10.

If you close a Container project, all subprojects close automatically. However, you can close one or more subprojects while keeping the Container project open. If you try to build a Container project where one of the subprojects is closed, this operation fails.

——— **Note** ———

The Container project defines the current working directory. This defines relative pathnames when you make changes to subprojects within the Container project.

10.3.3   Nesting Container projects

Container projects can be nested but not recursive, that is a Container project can include other Container projects but must not include itself. Nesting Container projects might be useful to help you organize your project files.

If you open a Container project that defines a Container subproject, the Project Control dialog box shows a second level in the open project list hierarchy.

Do not open a nested Container project if one of the subprojects is already open. Close the component project first so that the display is consistent.

10.4   Working with auto-projects

Any image that you load to a debug target must be accompanied by a project to save the image-related settings. If you open a user-defined project and this binds to a connection, this can be used to load the associated image. If you load an image directly, RealView Debugger searches for a saved auto-project file in the same location. If a saved file does not exist then RealView Debugger creates an in-memory auto-project to use in this session. You can save this file for later use.

If you load multiple images to the same target processor, the Process Control pane shows entries for the images and the auto-projects used to hold image-related settings, see the chapter describing working with images in *RealView Developer Kit v2.2 Debugger User Guide* for details on loading two images.

10.4.1   Working with user-defined projects and auto-projects

RealView Debugger enables you to work with multiple auto-projects or to mix user-defined projects and auto-projects in the same session:

*   If you are working with multiple projects in this way, normal binding rules apply, see *Types of binding* on page 9-2 for details.

*   If you display the Project Control dialog box, the open project list contains both auto-projects and user-defined projects. You can now work on both types of project in the usual way, see *Working with the Project Control dialog box* on page 6-3 for details.

*   It is not possible to distinguish between auto-projects and user-defined projects in the Project Control dialog box where they have the same name, unless the user-defined project is autobound, that is where (DEV) is appended to the project name. When you close one of the projects, use the Process Control pane to guarantee that you close the correct project, if possible. See *Displaying the Process Control pane* on page 6-5 for more details on these operations.

RealView Debugger enables you to view a combination of image details, auto-projects and user-defined projects in the Process Control pane, shown in Figure 10-3 on page 10-11.

                                   ARM DUI 0291C
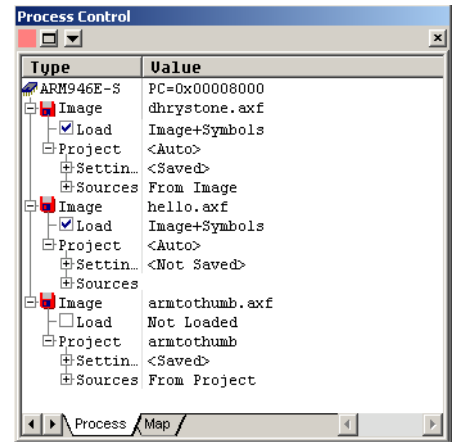
Figure 10-3 Mixing projects in the Process Control pane

Figure 10-3 shows:

- Two, non-overlapping, images loaded to the target processor where:

  — the first image, dhrystone.axf, is associated with a saved auto-project, shown by the entries Project <Auto> and Settings <Saved>

  — the second image, hello.axf, is associated with an in-memory auto-project, shown by the entries Project <Auto> and Settings <Not Saved>.

- The project settings file armtothumb.prj is open but is unbound. This means that it does not appear in the title bar of the floating Process Control pane.

- The image details associated with armtothumb.prj are registered with the debugger but the image is not loaded, shown by the entry Load <Not Loaded>.

- The active project is dhrystone.prj, shown in the default Code window title bar. This is reflected in the title bar of the floating Process Control pane.

However, the Project Control dialog box, shown in Figure 10-4 on page 10-12, indicates that there are other user-defined projects open in this session.
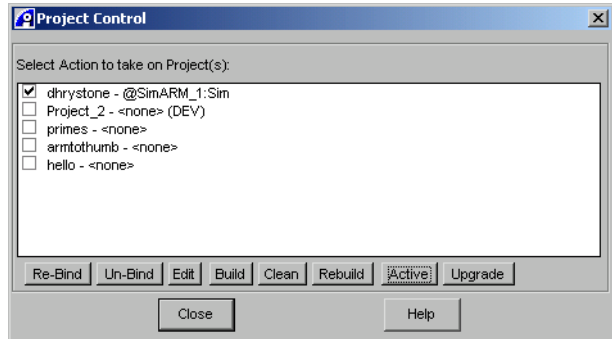
Figure 10-4 Mixing projects in the Project Control dialog box

Figure 10-4 shows, from the top:

- dhrystone. The auto-project associated with the loaded image (bound).

- Project_2. This user-defined project defines specific device binding, shown by (DEV). This is open and unbound.

- primes. This user-defined project is open and unbound.

- armtothumb. This user-defined project is open and unbound.

- hello. The auto-project associated with the second loaded image (unbound).

This example demonstrates the following rules:

- Unbound user-defined projects are not visible in the Process Control pane, for example Project_2 shown in Figure 10-4 is not visible in Figure 10-3 on page 10-11.

- If a user-defined project is bound to the connection, and you manually bind another user-defined project to the connection, then the process details for the second project replace the process details of the first project, in the Process Control pane.

- Auto-projects, associated with loaded images, are always visible in the Process Control pane, even where the auto-project is not bound to the connection, for example hello shown in Figure 10-3 on page 10-11 and Figure 10-4.

- It is not possible to distinguish between auto-projects and user-defined projects in the Project Control dialog box where they have the same name, unless the user-defined project is autobound, that is where (DEV) is appended to the project name (see Figure 10-4).

10.5   Closing projects

When you are working with multiple projects, you can close one or more projects as follows:

1.   Select **Project** → **Close Project** to see the open project list.

2.   Select the project(s) that you want to close.

3.   Click **OK**.

If you close a project that is bound to a connection, RealView Debugger uses the open project list to bind another project to the connection. The normal binding rules apply, see *Types of binding* on page 9-2 for details.

If you close a bound project that is also the active project, RealView Debugger sets the default active project:

•   The next project that binds to the connection becomes the active project.

•   If no project binds to the connection, the active project is the first project in the open project list.

If you close an unbound project that is also the active project, then the next project in the open project list becomes the active project by default.

――――― **Note** ―――――

When you close a project that is bound, RealView Debugger executes any close commands specified in the Commands_Open_Close group.

――――――――――――――――――

10.6   Connecting and disconnecting

Connecting to, and disconnecting from, a debug target changes the project environment:

- If you open multiple projects and then connect to a target, RealView Debugger binds any autobound projects first.

- If you open multiple projects where none specifies autobinding and then connect to a target, RealView Debugger uses default binding to bind projects in the order specified by the open project list.

- If you disconnect and there is a project bound to the connection, then the project unbinds but any close commands are not run because the connection has been lost. This clears the Process Control pane.

- If you disconnect, this might change the active project depending on the current project environment.

- If you disconnect, this does not close any open projects. This means that you can continue to make changes to the project properties. This applies to user-defined projects and auto-projects.

                   ARM DUI 0291C

Chapter 11
Editing Source Code

This chapter describes the file editing facilities that RealView® Debugger provides. It contains the following sections:

- *About the File Editor* on page 11-2
- *Configuring the File Editor* on page 11-4
- *Editing text* on page 11-5
- *Managing your editing session* on page 11-11
- *Using templates* on page 11-15.

11.1 About the File Editor

The File Editor provides a range of code editing features and enables you to specify your own personal editor if required. The File Editor enables you to:

- enter text to create project files
- open source files for editing and resaving
- edit binary files and save in a specified format
- use a range of text searching operations.

This section describes:

- *Using the File Editor pane*
- *Working with multiple files* on page 11-3
- *Editing binary files* on page 11-3.

11.1.1 Using the File Editor pane

The File Editor pane enables you to perform basic editing operations. Figure 11-1 shows two files loaded into the File Editor and displayed in the File Editor pane.
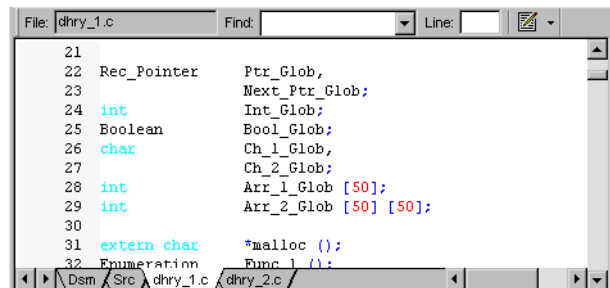
Figure 11-1 File Editor pane

Line numbers are not displayed by default but, in this example, this feature has been enabled. See *Configuring the File Editor* on page 11-4 for details of how to set this up for the current session.

The **File** field shows the name of the file currently displayed in the File Editor pane. If a file is *dirty*, that is if you have changed the file since loading or saving, an asterisk (*) is appended to the end of the filename.

The **Source control** button indicates the read/write status of the current file. Click this button to change the status of a file loaded into the File Editor. You can only edit the file if the Read-Write icon is displayed.

As each file is loaded into the File Editor, RealView Debugger creates a file tab in the File Editor pane. Click on the file tabs at the bottom of the File Editor pane to choose which file is currently visible. If a file has been edited, an asterisk (*) is appended to the front of the filename shown on the tab.

11.1.2 Working with multiple files

You can load source files into the File Editor using the menu option **File → Open...**. Multiple filenames can be highlighted in the File Chooser dialog and each file is loaded into its own tab.

You can create an empty tab in the File Editor. To do this, select **File → New** from the Code window main menu. This creates an empty tab, named <None>, ready to enter text from the keyboard or to paste text from another file.

11.1.3 Editing binary files

Be aware of the following if you are editing binary files in the File Editor:

- If you edit a binary file, the **Save** option is disabled. This means that your original file is safe from accidental corruption.

- If you open a binary file into the File Editor, the **Save As...** option is enabled. Select this option to save the updated file.

- The display format used in the File Editor pane means that you can edit any part of the file, including spaces. Be careful when saving a binary file that you have changed.

11.2　Configuring the File Editor

With a file visible in the File Editor pane, shown in Figure 11-1 on page 11-2, you can change display options. Select **Edit** → **Advanced** from the main menu to set up temporary File Editor display options. This displays the **Advanced Editing Controls** menu.

Select options from this menu to set:
- the default size for all tabs in the text
- the character width for Shift Lines Left and Shift Lines Right
- the text coloring scheme for source code
- automatic line numbering and renumbering
- tooltip evaluation for variables and registers.

Any changes you make here are temporary and are lost when you close down. Changes to line numbering and tooltip evaluation apply to the current Code window and do not persist to any new Code windows that you create. However, enabling text coloring or changing the default tab size or character width (for Shift Lines Left and Shift Lines Right) applies to the current Code window and to any new Code windows in the current session.

For more details on configuring your editing environment see *Managing your editing session* on page 11-11.

11.3 Editing text

The File Editor provides full editing options:

* *Basic text editing*
* *Formatting text* on page 11-7
* *Undoing changes* on page 11-9
* *Using drag-and-drop* on page 11-10.

11.3.1 Basic text editing

The File Editor supports the standard Windows editing operations.

Opening files

You can open saved files ready for editing in the File Editor:

* Select **File → Open...** from the Code window main menu. This displays a Select File to Open dialog box where you can locate the required file.

Closing files

You can save files after editing, for example:

* Select **File → Save** from the Code window main menu. This saves the current file using the original filename.

* Select **File → Save/Close Multiple...** from the Code window main menu. Select from the file list to save several files in a single step, shown in Figure 11-2.
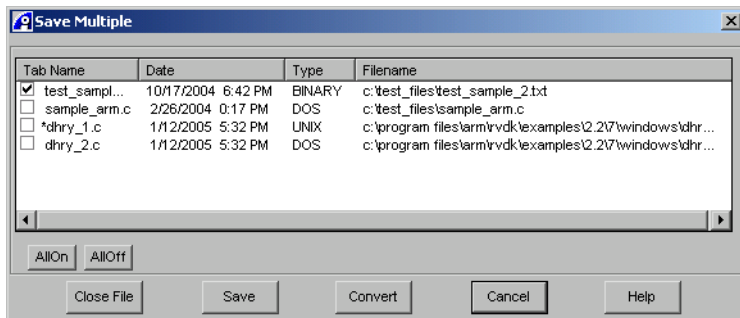
Figure 11-2 Save Multiple dialog box

The file displayed in the topmost tab of the calling window is checked by default ready to save. If a file has been edited, an asterisk (*) is appended to the front of the filename.

Select the files that you want to work on, or click **AllOn** to select all the files:

Close File   Closes selected files and removes their tabs from the File Editor pane. This closes the Save Multiple dialog box.

Save   Saves selected files and closes the dialog box.

Convert   Converts files from DOS to UNIX by removing end-of-line markers, or adds markers to convert UNIX files to DOS format:

- DOS files contain \r (carriage return) and \n (line feed)
- UNIX files contain \n (line feed).

Also converts a binary file to a text file (in the local format).

The Type column shows the current file type.

Cancel   Closes the Save Multiple dialog box without making any changes to the files.

Help   Displays online help.

RealView Debugger always warns if you select a file for closing that has been edited.

Adding text

To add text to an open file:

1. Click once in the File Editor pane to position the text insertion point.

2. Begin typing on the keyboard to enter text.

You can insert text from another file by copying the text block to the clipboard and pasting into your source file at the required position.

In addition, you can use a template to insert text into an open source file within the File Editor. See *Using templates* on page 11-15 for information.

———— **Note** ————

RealView Debugger warns if the destination file is read-only and asks for confirmation to enable you to edit the file.

Creating text

To add text to a new file:

1.   Select **File** → **New** from the Code window main menu. This creates an empty tab, called <None>, in the File Editor pane.

2.   Begin typing on the keyboard to enter text.

3.   Right-click on the tab to display the **File tabs** context menu.

4.   Select **Save File** to specify the new filename and location.

You can insert text from another file by copying the text block to the clipboard and pasting into your source file at the required position.

In addition, you can use a template to insert text into an open source file within the File Editor. See *Using templates* on page 11-15 for information.

Deleting text

You can delete text in any of the following ways:

•   Press the Backspace key to delete text that precedes the text insertion point.

•   Press the Delete key to delete text that follows the text insertion point.

•   Select the text you want to delete and press the Backspace or Delete key to delete the selection.

——— **Note** ———

RealView Debugger warns if the file is read-only and asks for confirmation to enable you to edit the file.

Working with file tabs

You can perform operations on text files using context menus available from the file tabs in the File Editor pane. For example, right-click on a file tab at the bottom of the File Editor pane and select **Close**. RealView Debugger warns you if the file has changed since the last save.

11.3.2   Formatting text

Where a file can be edited, you can format your source code to:

•   shift text left or right

- change case
- change capitalization
- enable source code coloring.

Use the **Edit** menu to set these options for the current session. To make the new setting the default when starting up the File Editor, you must change the Edit settings in your workspace (see *Changing your workspace settings* on page 11-12 for details).

For more details on formatting text see:

- *Shifting text left or right*
- *Enabling source code color*.

Shifting text left or right

To shift blocks of text left and right:

1. Select a block of text.

2. Select **Edit** → **Advanced** to display the **Advanced Editing Controls** menu.

3. Select either **Shift Lines Left** or **Shift Lines Right** as required.

The File Editor shifts the selected text a designated number of characters to the right or left by inserting or deleting spaces at the beginning of every line in the selection. The amount that text is shifted is determined by the value specified for the **Shift Width...** setting on the **Advanced Editing Controls** menu.

Enabling source code color

When working in the File Editor you can color source code as an aid to readability. This feature is enabled by default when you first start RealView Debugger. If source coloring is disabled, all text is the same color, usually black.

You can choose a color scheme from one of the following:

- Default
- Visual Studio
- CodeWarrior
- RealView Debugger v1.7.

                   ARM DUI 0291C

The defaults change depending on the scheme you choose. Table 11-1 shows the default colors used for each scheme if text coloring is enabled.

Table 11-1 Default text coloring for each scheme

Name	Default	Visual Studio	CodeWarrior	RealView Debugger v1.8
Numbers	Red	*System*[a]	Red	Blue
Strings	Magenta	*System*[a]	Magenta	Green
Keywords	Light Blue	Blue	Light Blue	Red
Comments	Green	Green	Green	Green
Identifiers	Blue	Blue	Blue	*System*[a]
User defined keywords	Blue	*System*[a]	Blue	Blue
Preprocessor keywords	Purple	Blue	Purple	*RVD17*[b]
Operators	Blue	*System*[a]	Blue	*System*[a]

a. This is set from your current system settings.
b. Yellow text on black background.

Standard C and C++ source coloring is auto-enabled based on file extension. The default list is defined in the Text settings in your workspace and can be changed if required. Specify file extensions as a comma-separated list which must not include dots or periods, for example c,cc,cxx,cpp,h,hpp.

11.3.3   Undoing changes

The File Editor provides several ways to undo mistakes as you edit a file.

Undoing edits

Select **Edit → Undo** to reverse the effect of your last action. The number of levels of undo is defined by the Undo value in the Edit settings in your workspace. The default value gives 64 levels of undo (see *Changing your workspace settings* on page 11-12 for details).

Undoing and redoing edits

Select **Edit** → **Redo** to redo your last undo. The number of levels of redo is defined by the Undo value in the workspace Edit settings (see *Changing your workspace settings* on page 11-12 for details).

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 **Re-Open File** from the file tabs context menu to return a file to the last-saved version. Because the current file is dirty you are warned to save changes before reloading.

11.3.4 Using drag-and-drop

You can use drag-and-drop to move selected text within the File Editor pane or to move text between File Editor panes.

A grayed-out box appears during the drag to represent the text being moved. The top-left corner of this box represents the starting point when text is inserted.

If you want to copy the text block rather than reposition it, hold down the Ctrl key as you drag the block. The grayed-out box is accompanied by a plus sign to show that this is a block copy.

Enabled by default, drag-and-drop text editing can be disabled in the Edit settings in your workspace. See *Changing your workspace settings* on page 11-12 for details of how to do this.

Moving text between windows

Each Code window can have only one instance of the File Editor pane. To drag and drop text between instances of the File Editor you must have:

- two or more Code windows open on the desktop
- files open in each Code window
- read/write access to any file into which you are moving text.

If the file that you are moving text from is set to read-only then the selected text is copied into the second window regardless of the action of the Ctrl key during the drag.

 ARM DUI 0291C

11.4   Managing your editing session

When you are working in the File Editor, there are different ways to configure your working environment:

- *Using Editing Controls*
- *Setting source search paths* on page 11-12
- *Changing your workspace settings* on page 11-12.

11.4.1   Using Editing Controls

Select **Edit** → **Advanced** from the Code window main menu to change editor settings for the current session.

Table 11-2 describes the options available from this menu.

Table 11-2 Advanced editing controls

Option	Usage
Shift Lines Left	Sets lines left by the number of characters specified by **Shift Width...**.
Shift Lines Right	Sets lines right by the number of characters specified by **Shift Width...**.
Shift Width...	Sets the number of characters by which text lines are shifted when formatting.
Show Line Numbers	Selected when line numbers are automatically displayed during editing. If enabled, lines are reordered as new lines are added or deleted.
Show Original Line Numbers	If selected, and line numbers have been enabled (see **Show Line Numbers**) line numbers are not changed as a file is edited. So, if new lines are added they have no numbers, and if lines are deleted there are gaps in the source file.
Reset Original Line Numbers	If selected, this option immediately resets line numbering to take account of line inserts and deletes.
Tab Size...	Sets the size of tab stops when editing source code. The default is 8 characters. The maximum value accepted is 16.
Auto Indent	Specify auto-indent for a new line when entering source code.
Text Coloring	Selected when text coloring is enabled for source code.
Tooltip Evaluation	Selected when tooltip evaluation of variables and registers is enabled.

Persistence

If you open a new Code window, RealView Debugger uses editor settings as defined by the current workspace, or the global configuration file if you are working without a workspace. Any changes that you make to these settings in the calling window do not, therefore, persist to the new window.

Exceptions to this rule are:
- Tab Size
- Shift Width
- Text Coloring.

11.4.2 Setting source search paths

The File Editor working directory is the location searched for a file when you do not give the full pathname. This is also the first place accessed by the debugger when searching for files.

If you have a project open you can select **Debug** → **Set Source Search Path...** to specify how path mapping is configured. For details on using this feature see *Specifying source search paths* on page 5-9.

11.4.3 Changing your workspace settings

You can change editor defaults by changing the Edit settings in your current workspace, shown in Figure 11-3, or in your global configuration file. Make changes here so that these settings are used the next time RealView Debugger starts and persist to all windows in a session.
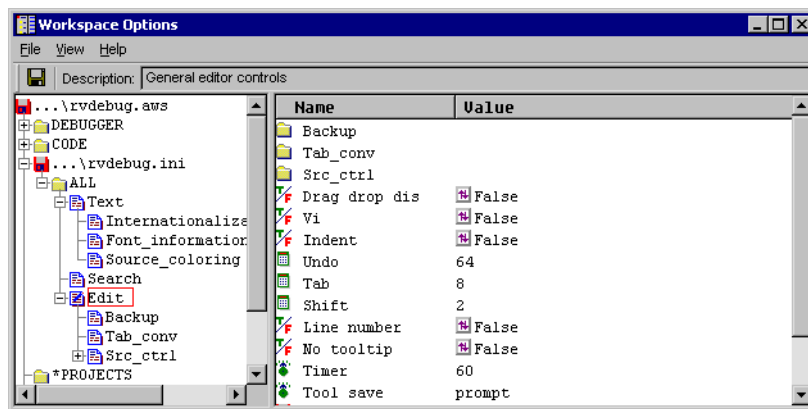
Figure 11-3 Edit settings in the workspace

If you are working with a workspace, select **File → Workspace → Workspace Options...** to make these changes. If you are working without a workspace, select **Tools → Options...** to make these changes to your global configuration file so that they are available in future debugging sessions.

Table 11-3 describes `Edit` settings that you can change.

Table 11-3 Edit settings in workspace

Name	Properties
Backup	File backup controls.
Tab_conv	Tab to space, and space to tab, conversion settings.
Src_ctrl	Source control settings.
Drag_drop_dis	Disable drag-and-drop text editing.
Vi	Not used.
Indent	Specify auto-indent for a new line when entering source code.
Undo	Specify the levels of undo and redo.
Tab	Specify tab spacing in the File Editor.
Shift	Specify the number of characters for Shift Left, Shift Right, and auto-indent.
Line_number	If set to `True`, this displays line numbers. The default is `False`.
No_tooltip	If set to `False`, the default, this displays hover-style evaluation when you hold your mouse pointer over a variable in the **Src** tab or a register in the **Dsm** tab.
Timer	The File Editor periodically checks to see if another editor has changed the current file. The default value is 60 seconds. You can enter a new value of >30 seconds. Setting this to -1 disables the check.
Tool_save	Tool-save enables the automatic saving of files in a build. The default is to prompt.
Startup	The default start-up file `rvdebug.sav` holds RealView Debugger information and the list of previously used editor files. Setting this to - disables this file.
Template	The default template file, see *Using templates* on page 11-15.
Restore_state	Not used.

For full details on changing your workspace settings to define editing controls, see the chapter describing configuring workspaces in *RealView Developer Kit v2.2 Debugger User Guide*.

11.5 Using templates

Templates are generic segments of text that you can modify to fit particular functions in your source code. You can use the File Editor to create templates or to modify existing templates ready to insert them into your source code.

——— **Note** ———

To insert a template in the current source file it must be editable. If it is not, click the **Source control** button to enable editing (see *Changing the status of controlled files* on page 13-3).

11.5.1 Inserting templates

A template can be inserted into an empty File Editor pane or added to existing code. The insertion point is determined by the location of the cursor.

By default, RealView Debugger loads the template file installed as part of the base product. This is called rvdebug.tpl, and is located in the default settings directory (\etc). However, you can create and use your own template files if required (see *Creating templates* on page 11-16).

To insert a template:

1. Open the file to be changed into the File Editor pane.

2. Position the text insertion point at the required location and select **Edit → Insert Template...**.

   This displays the Insert Template dialog box showing the currently available template file and the templates it contains, shown in Figure 11-4.
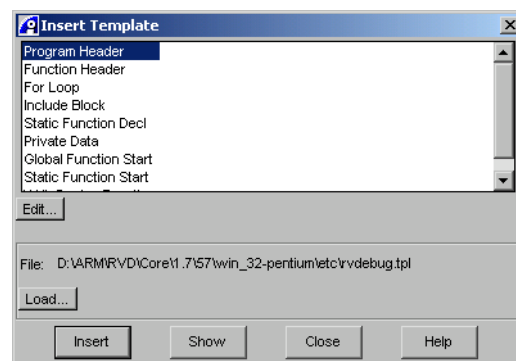
Figure 11-4 Insert Template dialog box

If you have created your own template file, use the **Load...** button to locate and load a different template file, for example `my_template.tpl`.

3. Highlight the required template in the list and click **Show** to display a text box showing the contents of the chosen template.

   Click **Close** to close the definition display.

4. Highlight the required template in the list and click **Insert** to paste the template into the current source code in the File Editor.

   The chosen template is displayed and any variables are automatically updated.

5. When all the required templates have been included, close the Insert Template dialog box and save your modified source code.

Editing templates

RealView Debugger provides several ways to edit template files:

• Edit template files from the Insert Template dialog box, shown in Figure 11-4 on page 11-15. Click **Edit...** to display the template file in the File Editor pane ready for editing.

• Load a template file directly into the File Editor pane without going through the Insert Template dialog box.

• Create and edit template files using any text editor.

11.5.2 Creating templates

You can create your own templates, or modify an existing template, in an editor and then save the newly-created file. When saving your own templates you must use the `.tpl` extension.

A template takes the format:

```
"templatename" TEMPLATE [-above|-below|-indent|-selection]
             [-auto=<phrase>] [filename]
.
.
.
"templatename" END
```

where:

`-templatename`

Specifies the template name.

-above	Use this to insert the template on the line above the current cursor.	
-below	Use this to insert the template on the line below the current cursor.	
-indent	Templates are indented to match the previous line.	
-selection	Indicates that the template contains variables that are replaced on insertion into the source code. Valid variables are:	

	$selection	The currently selected text.
	$paste	The current paste buffer.
	$filename	The current filename, without the path, in lowercase.
	$FILENAME	The current filename, without the path, in uppercase.
	$directory	The path without the filename.

-auto=<*phrase*>

Specifies the auto insertion *phrase*. The *phrase* is a name associated with a defined template. When a template has an auto insertion phrase, there are three actions available after typing in the *phrase*:

Ctrl+Shift+9	Insert the template at the cursor position.
Ctrl+9	View the definition of the template specified by *phrase*.
Alt+9	Invoke the Insert Template dialog.

filename   Specifies the file containing text to define the template. Where no file is given, the lines following the template definition header are used until the line "templatename" END is reached.

Insertion markers

Within a template, you can place a ^L (ASCII 12) character after a comment section. The lines immediately preceding the ^L are not inserted.

Nesting templates

You can also nest templates within templates using the syntax:

"filename" INCLUDE

11.5.3   Template examples

The following examples demonstrate how to create templates and describe different ways to insert templates into source code.

Each example can be typed into a .tpl file or they can all be entered into a single file, for example template_examples.tpl, for access through the Insert Template dialog box.

Example 11-1 shows a template file using text from the specified file, called `myfile.txt`. On insertion, the text is placed above the cursor position.

Example 11-1

```
# Simple template example using text from a specified file
"TemplateName" TEMPLATE -above C:\test_files\myfile.txt
"TemplateName" END
```

Example 11-2 shows a template of a simple FOR loop written in C. On insertion, the text is placed below the cursor position and indented to match the line of C code that immediately precedes it.

Example 11-2

```
# Simple template example - simple FOR loop
"Simple For Loop" TEMPLATE -below -indent
for (index = 0; index < top; index++);
{
}
"Simple For Loop" END
```

Example 11-3 shows a template that uses an auto-insertion phrase and demonstrates the use of the ^L (ASCII 12) character. With this template, type the auto-insertion phrase `WriteDevice` and immediately press Ctrl+Shift+9 to insert the template. Only the text immediately after the ^L is inserted.

Example 11-3

```
# Simple template example
# This combines an auto insertion phrase and the use of ^L (ASCII 12)
"WriteDevice Function" TEMPLATE -auto='WriteDevice'
EXAMPLE
/* write a message string to the LCD */
if (!WriteDevice(main_lcd, sizeof(init_msg), init_msg))
    ReportFailure(DEV_ERR, class_lcd, main_lcd);
```

```
^L
Status WriteDevice(int dev_no, int len, uint8 *buffer);
"WriteDevice Function" END
```

Example 11-4 shows the use of selection when inserting a template. If you are editing source code and you select a function name within a definition and insert this template, the body is expanded above the function and the function name is filled in where $selection is found.

Example 11-4

```
# Simple template example using selection
"Function Header" TEMPLATE -sel -above
/*
--------------------------------
$selection -
Notes:
--------------------------------
*/
"Function Header" END
```

Chapter 12
Searching and Replacing Text

This chapter describes how to use RealView® Debugger search and replace functions. It contains the following sections:

12.1   About finding and replacing text

The find and replace functions in the File Editor enable you to:

- work within the current Code window
- work between windows
- work across different windows.

In search and replace operations you can use:

- text strings
- pattern matches
- grep-style regular expressions.

       ARM DUI 0291C

12.2 Searching and replacing text

This section describes how to use the search features to find specific text within a file in the File Editor pane. See:

- *Searching for text*
- *Searching and replacing text* on page 12-5
- *Searching for selected text* on page 12-6.

——— **Note** ———

If you have enabled Internationalization, then you can also search for multibyte text in the selected encoding format.

12.2.1 Searching for text

To quickly find a text string in the current source file displayed in the File Editor pane:

1. Enter the required text string into the Find data field above the window.

2. Press the Enter key to find the first occurrence.

3. Press Enter again to find the next occurrence.

As you enter a search string, RealView Debugger copies the string to the clipboard. This means that the most-recently used string is available when you next display the Find, or Find and Replace, dialog box.

To find a text string in the current source file using the Find dialog box:

1. Position the cursor in the current file.

2. Select **Edit** → **Find...** to display the Find dialog box, shown in Figure 12-1.
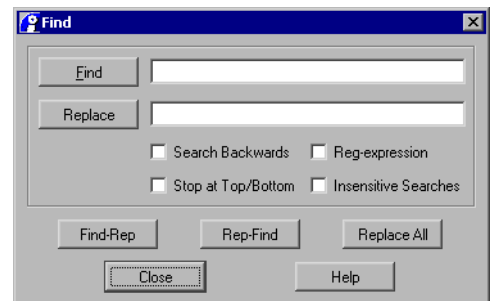
Figure 12-1 Find dialog box

3. Enter the required text string into the Find field in the dialog box.

The last search string used might already be displayed if available from the clipboard.

4.   Click the **Find** button.

5.   When a match is highlighted either:

   •   click the **Find** button again to search for the next occurrence of the text string

   •   replace the current selection, see *Searching and replacing text* on page 12-5.

Table 12-1 describes the options available from the Find dialog box.

Table 12-1 Setting temporary search settings

Option	Usage
Search Backwards	By default, a file is searched forwards, that is from the current cursor position to the end of the file. Set this option if you want to reverse the search direction. This might be useful if you miss a previous match and want to retrace your steps to verify the occurrence or to make a replacement.
Reg-expression	By default, the File Editor assumes that the string in the Find field is a text string. Set this option to force the editor to interpret the string as a regular expression.
Stop at Top/Bottom	By default, any search operation cycles through the current file and returns to the starting point. Set this option to force the search to stop when it reaches the top or the bottom of a file.
Insensitive Searches	By default, any search is case sensitive. That is uppercase and lowercase are distinct so searching for ARM does not find Arm or arm. Setting this option means that uppercase and lowercase are treated as identical.

When the first match is found, you can move the focus to the File Editor pane and press F3 to find the next occurrence of the text string. It is not necessary to close the Find dialog box.

───── **Note** ─────

See *Configuring searches in the workspace* on page 12-16 for details on how to configure the Search settings in your workspace.

Searching multiple tabs

If you display two or more file tabs in the current Code window then the search is restricted only to the file tab that is currently visible. However, you can select another file tab without closing the Find dialog box to continue the search.

12.2.2 Searching and replacing text

To replace text in the current source file it must be editable. If it is not, click the **Source control** button to enable editing (see *Changing the status of controlled files* on page 13-3).

To find and replace text in the current source file displayed in the File Editor pane:

1.   Position the cursor in the current file.

2.   Select **Edit → Replace...** to display the Find and Replace dialog box. This has the same controls as the Find dialog shown in Figure 12-1 on page 12-3.

3.   Enter the required text string into the Find field.

4.   Click the **Find** button.

5.   If there is no occurrence of the search string in the current file then an information box is shown saying that there is no match, the default, or the Code window flashes.

6.   When a match is highlighted either:

   •   click the **Find** button again to search for the next occurrence of the text string

   •   replace the current selection.

7.   Click **Close** to close the Find and Replace dialog box.

Making global changes

The Find and Replace dialog box also enables you to specify a global change where every occurrence of the search string is replaced in a single operation. Use the **Replace All** button to make a global change.

The search stops when all the matches have been replaced. There is no report detailing the number of replacements made but searching for the same string again shows that it no longer exists in the current file.

If you want to make the same global change in a second file tab then switch to the new tab without closing the Find and Replace dialog box and click **Replace All** again. In this way all occurrences of the text string are replaced in the second file.

Case sensitivity

By default, searches are case sensitive, that is, uppercase and lowercase are treated as distinct. Select the **Insensitive Searches** check box on the Find or the Find and Replace dialog box to change this for the current debugging session. When replacing a text string, the replacement string is always used exactly as entered in the Replace field. You must also enter spaces if required, in both the Find and Replace fields, because these are matched exactly.

Undoing replacements

You can reverse any search and replace operation. To do this, select the **Edit** → **Undo** option to reverse the last action. Where you have used the **Replace All** button, you must undo each replacement in turn as there is no **Undo** option for a global replacement.

12.2.3 Searching for selected text

To search for selected text, highlight the required search text in the current file tab, then do one of the following:

- To search for the selected text in the file currently displayed in the File Editor pane:

    1. Select **Edit** → **Find...** from the main menu. The Find dialog box is displayed with the selected text shown in the Find field.

    2. Click the **Find** button. The next occurrence of the selected text is highlighted. Repeat this step to find each remaining occurrence of the selected text. See *Searching for text* on page 12-3 for instructions on using the Find dialog box.

- To search for the selected text in all files in a specific location, select **Edit** → **Find in Files...** from the main menu. The Find in Files dialog box is displayed with the selected text shown in the Search For field. See *Searching multiple files* on page 12-7 for more details.

12.3   Searching multiple files

This section describes how to use the Find in Files dialog box to:

*   search for a text string across multiple files
*   search for a text string by specifying how it is used in the code.

This dialog box searches for matches in files already stored on the disk. If you are editing a file in the File Editor that has not been saved to disk and a match is found in this file, then line numbers and the position of the indicator might not be shown accurately in the file tab. Therefore, to ensure that matches are located correctly, you must save any files before starting the search.

———— **Note** ————

You cannot search for multibyte text across multiple files.

It includes:

*   *Displaying the Find in Files dialog box*
*   *Components of the Find in Files dialog box* on page 12-8
*   *Abandoning the search* on page 12-9
*   *Viewing search results* on page 12-9.

12.3.1   Displaying the Find in Files dialog box

Select **Edit → Find in Files...** to display the Find in Files dialog box, shown in Figure 12-2.
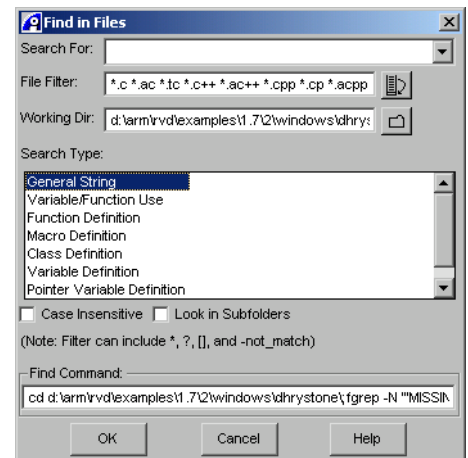
Figure 12-2 Find in Files dialog box

12.3.2 Components of the Find in Files dialog box

Enter data into the following fields to specify the search:

Search For: Enter the search string. Later you can limit the search by specifying how this string is used in the source file.

File Filter: Use this field to specify the file types included in the search. You can limit the search to files having a particular extension or to files starting with a chosen letter. The file filter can also contain regular expressions built with the operators ?, *, [], and -not_match, for example [ms]*.c or [m?s?]*.[ch]. Click the **File Filter** button to display a list of previously used filters.

Working Dir:

This field contains the pathname of the default working directory which is the starting point for the search. You can click the **Working Directory** button to see a list of previously used pathnames to change this entry.

Search Type:

You can limit the search by specifying how the search string is used in the code. Select the usage you require from the Search Type list:

- **General String** searches for a text string regardless of how it is used.

- **Variable/Function Use** looks only for a string used as a variable or function parameter.

- **Function Definition** finds strings in a function definition but does not find calls to the function.

- **Macro Definition** finds strings in a macro definition but does not find calls to the macro.

- **Variable Definition** searches for strings in a variable definition but does not match uses of the variable.

- **Class Definition** searches for strings in a class definition but does not match uses of the class.

- **Pointer Variable Definition** finds strings in a pointer variable definition. It does not match uses of the pointer variable nor does it find places where the variable was defined but not as a pointer.

- **Typedef Definition** finds only strings that match the defined type.

Case Insensitive

Makes any searches case insensitive, that is treating uppercase and lowercase as identical. This is independent of the status of the check box on the Find, or the Find and Replace, dialog box.

Look in Subfolders

Includes subfolders when searching the working directory.

Find Command

As you set up the search criteria, the Find Command field shows the grep-style command that RealView Debugger uses to perform the search. You can manually edit this command in this field. When the command is complete, click **OK** to start the search.

12.3.3 Abandoning the search

If you want to abandon the search, you can:

*   click the **Cancel** button and close the Find in Files dialog box
*   click the **Stop Build/Compile/Find** button on the main toolbar
*   select **Build → Stop Build/Find**.

12.3.4 Viewing search results

The results of any **Find in Files** search are displayed in two ways:

*   a summary of matches is shown in the **FileFind** tab in the Output pane
*   matching files can be displayed in the File Editor.

Viewing matches in the FileFind tab

Click the **OK** button in the Find in Files dialog box to make the **FileFind** tab current in the Output pane and display the grep -f command being executed.

If the search finds matches then the results are displayed in the **FileFind** tab.

If the search finds matches then the results are displayed in the **FileFind** tab where the first matching occurrence of the search string is highlighted. Each entry shows:

*   the filename in the chosen search path where a match has been found

*   the line number in the file where the match has been found, set up by default using the -N flag

*   the contents of the matching line.

Double-click on a chosen result line to move through the results list shown in the **FileFind** tab.

Viewing matches in the File Editor

If the current File Editor is empty, or you are editing a file not included in the results list, then a successful search creates a new file tab in the File Editor pane. This contains the first file in the search results and you can see a blue indicator pointing at the first matching line in the file. The matching line of code is also highlighted in the **FileFind** tab in the Output pane.

If you move down to a match in the results list and the corresponding file is not currently displayed in the File Editor then a new file tab is created automatically and the indicator positioned at the matching line.

If the File Editor already contains file tabs for all the files where successful matches have been identified then the corresponding tabs are displayed as you move through the results list in the Output pane.

 ARM DUI 0291C

12.4 Working with functions

You can use the **Edit → Go to** menu options to search for, and move between, functions when editing your source files:

- *Jumping to a function*
- *Jumping to a function definition* on page 12-12
- *Returning from a jump* on page 12-12.

———— **Note** ————

To use the Go to menu options, the current source file must be editable. If it is not, click the **Source control** button to enable editing (see *Changing the status of controlled files* on page 13-3).

12.4.1 Jumping to a function

Select **Edit → Go to → Jump to Function...** to display the Jump to Function dialog box to see a list of all the functions in the source file shown in the current file tab, shown in Figure 12-3.
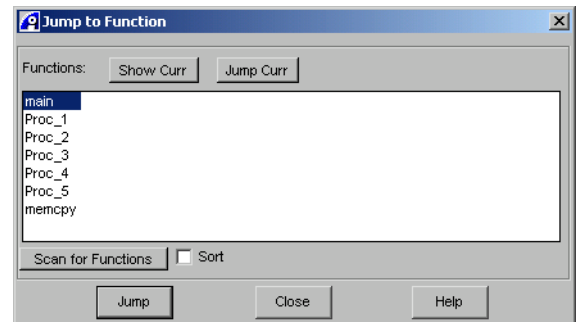
Figure 12-3 Jump to Function dialog box

By default, the functions list is displayed sequentially, that is in the order the functions occur in the source file. Click the **Sort** check box to re-order the functions list into alphabetical order.

If you select another file tab, in the File Editor pane, and then click **Scan for Functions** the list is refreshed for the new source file.

Click **Jump**, or double-click on the required name, to locate a function in your source file. This scrolls the file in the File Editor pane so that the first line of the function is visible towards the top of the pane. Click **Close** to close the Jump to Function dialog box and place the cursor at the start of the function ready for editing.

When editing a large file, the Jump to Function dialog is useful to find your way around. Select **Edit → Go to → Jump to Function...** to display the Jump to Function dialog box containing the functions list for the current source file and the controls:

Show Curr   Moves the functions list highlight bar to show which function currently contains the cursor.

Jump Curr   Scrolls the file in the File Editor pane so that the first line of the function, currently containing the cursor, is towards the top of the pane. Click **Close** to close the Jump to Function dialog box and to place the cursor at the start of the function ready for editing.

12.4.2 Jumping to a function definition

If you are editing a large file in the File Editor, it is useful to be able to move to a function definition from the function call. In Example 12-1 the cursor is located at the call to Func_1. Select **Edit → Go to → Jump to Function/Include at Cursor** to move the cursor to the start of the definition of the function.

Example 12-1 Function call example

```
Int_Loc = 2;
    while (Int_Loc <= 2)
    if (Func_1 (Str_1_Par_Ref[Int_Loc],
        Str_2_Par_Ref[Int_Loc+1]) == Ident_1)
```

12.4.3 Returning from a jump

If you have used **Jump to Function...** or **Jump to Function/Include at Cursor** to jump to a function or function definition in the current source file, you can jump back to the starting point.

Select **Edit → Go to → Jump Back** to reverse the last jump and return the cursor to the location before the jump was made. Unlike the **Edit → Undo** option, the maximum number of return jumps you can make is fixed at 16.

12.5 Pair matching

You can use the File Editor to search for, and move between, structures when editing your source files. Using this feature enables you to search your source code for:

- structure definitions consisting of a list of declarations enclosed in braces, that is curly brackets or {}

- function definitions containing a list of parameter declarations enclosed in parentheses, that is round brackets or ()

- array declarations enclosed in square brackets or []

- characters enclosed by double quotes, for example printf("\n");

- character constants enclosed by single quotes, for example if (c == '\n')

- comments

- conditions specified by if/else

- conditions evaluated during preprocessing specified by #if/#else/#endif.

This section contains:
- *Searching pairs*
- *Matching pairs* on page 12-14
- *Matching nested pairs* on page 12-15
- *Enclosing* on page 12-15.

12.5.1 Searching pairs

Select the option **Edit → Go to → Pair Matching...** to display the Pair Matching dialog box, shown in Figure 12-4. Use this to search for pairs in the current file tab.
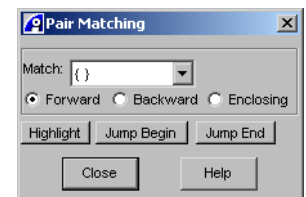
Figure 12-4 Pair Matching dialog box

Specify the type of structure you are looking for in the Match field. Click on the drop-down arrow to display the pair selection list where you can highlight the required pair delimiters.

Configuring the search

The current cursor position defines the starting point but you can specify how the search is carried out:

- select the **Forward** radio button to search from the starting point to the end of the file

- select the **Backward** radio button to search from the starting point to the beginning of the file

- select the **Enclosing** radio button to look for the pair currently enclosing the cursor, see *Enclosing* on page 12-15 for more information.

If you have enabled wrapping during search operations, this does not affect searching for pairs. When the end or beginning of the file is reached the search stops and reports no more matches using an information box (the default option).

12.5.2 Matching pairs

Select **Edit → Go to → Pair Matching...** to display the Pair Matching dialog box shown in Figure 12-4 on page 12-13. This dialog box contains:

Highlight   Begins the search and scrolls the file to display the first match. The code within the matching pair is then highlighted in the File Editor pane. The search locates the first delimiter in the matching pair and places the cursor after it.

Jump Begin   Begins the search and scrolls the file to display the first match. The search locates the first delimiter in the matching pair and places the cursor after it.

Jump End   Begins the search and scrolls the file to display the first match. The search locates the second delimiter in the matching pair and places the cursor after it.

When a pair is matched in your source code, the File Editor displays the result by highlighting the pair or repositioning the cursor. When moving the cursor, the new location is displayed in the Cursor location field. You can make the cursor visible by making the Code window active.

Where you have more than one matching condition in your source file, you can continue searching through the file until you reach the end. This displays an information box or flashes the Code window to show that no more matches have been found.

12.5.3   Matching nested pairs

You can search for pairs in source files containing nested structures in the same way. However, the way pairs are matched depends on the:

- starting point
- search direction, that is whether you select **Forward** or **Backward**.

12.5.4   Enclosing

When configuring your search, select the **Enclosing** radio button, from the Pair Matching dialog box, to find the nearest pair delimiters enclosing the cursor.

If the cursor is not inside a pair of the selected type when the **Enclosing** radio button is selected, then the No match information box is displayed or the Code window flashes.

12.6   Configuring searches in the workspace

You can change search defaults by changing the Search controls in your current workspace or in your global configuration file, shown in Figure 12-5. Make changes here so that these settings are used the next time RealView Debugger starts up.
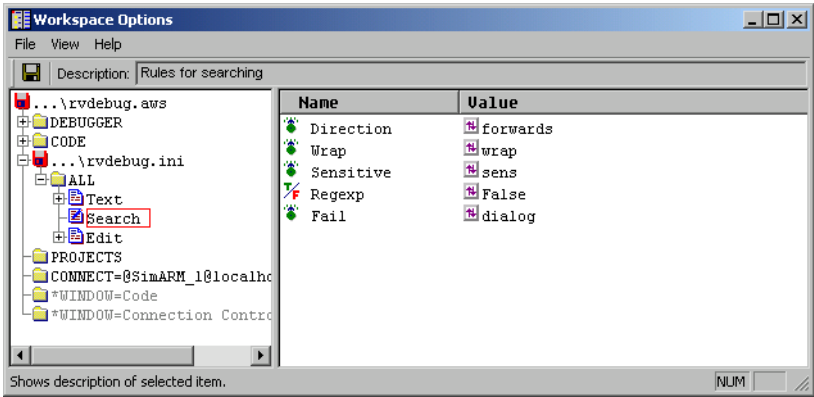
Figure 12-5 Search settings in the workspace

If you are working with a workspace, select **File → Workspace → Workspace Options...** to make these changes. If you are working without a workspace, select **Tools → Options...** to make these changes to your global configuration file so that they are available in future debugging sessions.

Table 12-2 describes Search settings that you can change.

Table 12-2 Search settings in workspace

Object	Properties
Direction	The direction followed by a search. The default is to search forwards, that is from the top to the bottom of the file. To change this, set to backwards.
Wrap	The default is to wrap during a search, that is to search to the end of the file and then to start again at the top until the starting point is reached. To change this, set to stop.

Table 12-2 Search settings in workspace

Object	Properties
Sensitive	This determines whether uppercase and lowercase are treated as identical in searches. The default is case sensitive. To change this, set to `insens`.
Regexp	When set to `T`, or `True`, full grep-style regular expressions are used in searches. The default is `False`, that is not enabled.
Fail	When a search fails to find a match, the File Editor can display an information box or flash the File Editor pane. Set to `dialog` by default, you can change this to `flash`.

For full details on changing your workspace settings to define search controls, see the chapter describing configuring workspaces in *RealView Developer Kit v2.2 Debugger User Guide*.

12.7 Using regular expressions

The File Editor provides regular expression matching that is similar to the Sun Solaris grep command. A regular expression is a text string composed of characters, some of which have special meanings. Therefore, the regular expression string used in the search describes one or more possible literal strings. Similarly, you can use regular expressions to both search for and replace strings in your source files.

Using regular expressions enables you to carry out complex searches when editing your source files. This section gives a brief introduction. For a comprehensive reference guide to regular expressions, see *Mastering Regular Expressions* edited by Jeffrey E.F. Friedl.

This section contains:

- *About special operators*
- *Searching with regular expressions* on page 12-19.

12.7.1 About special operators

Table 12-3 shows the characters that have special meanings in a regular expression. In some cases, their meaning depends on where they occur in the regular expression.

Table 12-3 Regular expression metacharacters

Metacharacter	Description
.	The *match-any-character operator* matches any single character.
*	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 refers to a literal character within the regular expression.
|	The *alternation* operator matches one of a choice of regular expressions, for example REG|Glob. If you place the alternation operator between any two regular expressions, the result matches the largest union of strings that it can match.

Table 12-3 Regular expression metacharacters

Metacharacter	Description
^	The *match-beginning-of-line* operator matches the string from the beginning of the string or after a new-line character, for example ^REG.
^	The *not* operator is used inside square brackets, or [], to represent a NOT action, for example R[^EG].
$	The *match-end-of-line* operator matches the string either at the end of the string or before a new-line character in the string, for example Comp;$.
[...]	*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, for example [Val].

12.7.2   Searching with regular expressions

By default, text searches do not assume regular expressions. You can choose to set up your searches so that regular expressions are assumed, either:

- change the Search controls in your current workspace
- select the **Reg-expression** option in the Find (and Replace) dialog box.

Use the second method to configure regular expression matching for the current session only. When set, displaying the Find, and Find and Replace, dialog box opens with the radio button selected. You must change your workspace settings to enable this feature for all sessions.

To enable regular expression searches for the current File Editor session:

1.   Select **Edit → Find...** or select **Edit → Replace...** to start a search sequence and replace matched text.

2.   Select the **Reg-expression** option on the Find (or Find and Replace) dialog box.

3.   Enter the required expression in the Find field.

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

Matching simple expressions

Most characters in a regular expression match themselves. For example, entering the regular expression struct in the Find field matches all occurrences of the string struct in your source file. This changes, however, when the regular expression contains metacharacters as listed in Table 12-3 on page 12-18.

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 field. The backslash instructs the File Editor to interpret the dollar sign as a literal character, rather than a special character. If you do not use the backslash, the search finds the end-of-line characters instead.

Matching any single character

Using a dot or period (full stop) matches any single character so entering the regular expression var. in the Find field matches any four character sequence beginning with var, such as var1, and var2, and var followed by a space. It does not match var followed by an end-of-line character.

Matching alternative expressions

Using an alternation operation or pipe (|) matches alternative expressions so entering the regular expression REG|Glob in the Find field matches either REG or Glob. This can also be combined with the *not* operator, for example [^REG|Glob] which applies to both REG and Glob.

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 none, one, or more occurrences of that regular expression

- a regular expression followed by a plus sign, +, matches one or more occurrences of that regular expression

- a regular expression followed by a question mark, ?, matches none or one occurrence of that regular expression.

Table 12-4 describes some simple examples to illustrate matching repeating expressions.

Table 12-4 Using repetition operators

Regular expression	Matches
s*ion	None, one, or more occurrences of the s character immediately preceding the characters ion. This regular expression matches, for example, with ion in information, sections, and expressions, and with ssion and sion in expressions.
s+ion	One or more occurrences of the s character immediately preceding the characters ion. This regular expression matches, for example, the ssion and sion in expressions.
s?ion	None or one occurrence of the s character immediately preceding the characters ion. This regular expression matches, for example, with the sion in expressions, and with ion in information, sections, and expressions.
0\.?	The number zero, followed by a dot or period (full stop). The backslash tells the File Editor to treat the dot as a literal character, and the ? operator acts on the dot. This regular expression matches, for example, with 0. and 0 followed by a character or an end-of-line character.

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

Grouping expressions

If an expression is enclosed in parentheses or round brackets (), 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 the text string is in the Find field. However, you can also use (i)s as a regular expression. This instructs the File Editor to look for the letter s, preceded by both a space and the letter i. So, while using the text string search is matches with This, this, and is, the regular expression (i)s matches 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 or *not* operator (^). 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 characters, use a minus sign (-) between the start and end characters, and place the whole expression within square brackets. For example, the regular expression [0-9] is the same as [0123456789].

The following applies 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 characters -, b, and c.

- 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 dot or period (full stop).

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 while [aeiou][0-9] matches any lowercase vowel followed by a number, such as a1.

Matching the start or end of a line

You can use a regular expression to search for start-of-line and end-of-line characters:

- If a caret, ^, is at the beginning of the entire regular expression, it matches the beginning of a line. For example, the regular expression ^reg_opt matches any occurrence of reg_opt but only at the start of a line.

- If a dollar sign, $, is at the end of the entire regular expression, it matches the end of a line. For example, reg_opt$ matches any occurrence of the string reg_opt but only at the end of a line.

- If an entire regular expression is enclosed by a caret and dollar sign (^par_a4 == reg_opt$), it matches an entire line.

You can build complex search strings by combining regular expressions and metacharacters, for example ^([aeiou][0-9]) or ([aeiou][0-9])$.

Chapter 13
Working with Version Control Systems

This chapter describes RealView® Debugger support for version control software to manage access to source files and other files associated with debugging your applications, such as user-defined projects. It contains the following sections:

13.1 Defining the version control tool

When you first run RealView Debugger after installation (or without a Workspace defaults file), it attempts to identify your version control tool:

1. Is ClearCase specified in the Registry?

2. Is PVCS specified in the Registry and the PCVS executable on the path?

3. Is the environment variable CLEARCASE_ROOT set?

4. Is the environment variable CVSROOT set?

If all checks fail then RealView Debugger assumes that you do not have a version control system.

If RealView Debugger fails to identify your version control tool, you must specify it. See *Configuring a custom version control tool* on page 13-8 for details on how to do this.

––––––– **Note** –––––––

RealView Debugger continues to use the specified version control tool until you change it explicitly.

–––––––––––––––––––––––

13.2 Using a version control tool

This section describes how to manage source files using version control commands. In these examples, RealView Debugger is running on a Windows workstation using WinCVS to manage source files.

——— **Note** ———

RealView Debugger has been configured to recognize the version control software because it was not detected automatically. The Workspace settings file has been updated as described in *Configuring a custom version control tool* on page 13-8.

This section describes:

- *Changing the status of controlled files*
- *Using source control commands*
- *Setting up a prompt* on page 13-7.

13.2.1 Changing the status of controlled files

Opening a file that is under version control, activates the **Source control** button on the Editing toolbar. This button shows the read/write status of the current file:

Locked Indicates that the file is locked.

A file marked in this way cannot be edited without changing the status.

Read-Write Indicates that the file is editable.

A file marked in this way can be edited without changing the status.

Read-only Indicates that the file is read-only.

A file marked in this way cannot be edited without changing the status.

If a file is read-only in the File Editor pane, and you try to edit, RealView Debugger displays a prompt, where you can choose to change the file status so that editing can take place.

Click **Yes** to change the file status so that it can be edited. This changes the icon displayed on the **Source control** button. You can also change the status of the file by clicking on the **Source control** drop-down arrow to display the **Source control** menu.

13.2.2 Using source control commands

Click the **Source control** button drop-down arrow to access the source control commands. The options available from this menu depend on the status of the file loaded into the File Editor pane and the version control tool you are using.

Locked, read-only files

If you are working with a locked file, click the **Source control** button drop-down arrow to display the **Source control** menu shown in Figure 13-1.

| <Not Editable> |
| Raise WinCVS |
| Prompt before Submitting |
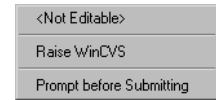
Figure 13-1 Source control menu for locked files

The menu options are:

<Not Editable>

        Indicates the status of the file as Locked and so it cannot be edited.

Raise WinCVS

        Starts up the version control tool.

Prompt before Submitting

        Source control commands chosen from the **Source control** menu are submitted directly for execution. Click this option to display a prompt showing the command before submission, see *Setting up a prompt* on page 13-7 for details.

If you are working with a read-only file, the **Source control** menu includes:

Allow Editing

        Click to change the status of the file to Read-Write so that it can be edited.

        RealView Debugger warns if the contents of the buffer have changed since the file was last saved.

Read-Write files

If you are working with an editable file, click the **Source control** button drop-down arrow to display the **Source control** menu shown in Figure 13-2.

| Make Buffer Read-Only |
| Check-In File |
| UnCheck-Out File |
| Enter File into Source Control |
| Raise WinCVS |
| Show Files You have Checked Out |
| Show Version History |
| Show Changes from Previous Version |
| Show Differences of two Versions |
| Show Tags |
| File Status |
| Merge Changes |
| Prompt before Submitting |
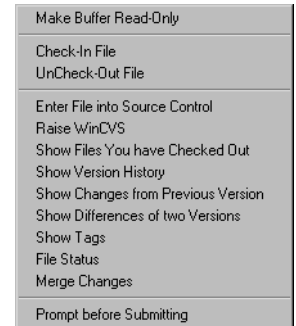
Figure 13-2 Source control menu for read-write files

In this example, the menu options include:

Make Buffer Read-Only

Changes the status of the file to Read-only so that it cannot be edited.

Check-In File

Checks the current file into the repository. Selecting this option displays a message dialog box where you can enter a text comment, for example Updated for new tools env. This is used as the message to identify the check-in, for example:

cvs commit -m "Updated for new tools env" *filename*

UnCheck-Out File

Updates the current file to synchronize it with any changes saved to the repository since it was last checked out or committed, for example:

cvs update *filename*

The original file is renamed, for example dhry_1.c.old.

Enter File into Source Control

Adds the current file to the repository, for example:

cvs add *filename*

Raise WinCVS

Starts up the version control tool.

Show Files You have Checked Out

Updates the current file to merge changes. Where a file contains changes, this is roughly equivalent to a `checkout` command, for example:

`cvs -nq update`

This does not complete the update, defined by the `-n` flag. It is also quiet (`-q` flag).

Show Version History

Displays the history information for the current file, for example:

`cvs -q log -N filename`

This does not list tags, defined by the `-N` flag. It is also quiet (`-q` flag).

Show Changes from Previous Version

Displays differences between the current file and the previous version in the repository, for example:

`cvs diff filename`

Show Differences of two Versions

Displays differences between two versions of the file currently checked out. Selecting this option displays the first of two prompts where you specify the version numbers to compare, for example:

`cvs diff -r 1.1 -r 1.3 filename`

Show Tags   Displays status information for the current file, including tags, for example:

`cvs status -v filename`

File Status   Displays status information for the current file, for example:

`cvs status filename`

Merge Changes

Updates the current file to merge changes. Where a file contains changes this is equivalent to a `checkout` command, for example:

`cvs update filename`

This does complete the update.

If you are working with a custom version control tool, that is one that is not supported by RealView Debugger by default, you can specify the commands submitted. These appear as options on the menus described in this section. See *Specifying custom commands* on page 13-10 for details on setting up these custom commands.

13.2.3   Setting up a prompt

By default, source control commands are submitted directly for execution. This is defined in your workspace settings file. You can configure this:

- change the `Src_ctrl` settings in your workspace, described in *Configuring a custom version control tool* on page 13-8

- set up a temporary prompt.

To set up a temporary prompt, select **Prompt before Submitting** from the **Source control** menu, shown in Figure 13-2 on page 13-5. From this point on, RealView Debugger displays the prompt, shown in Figure 13-3, to confirm a command submission.
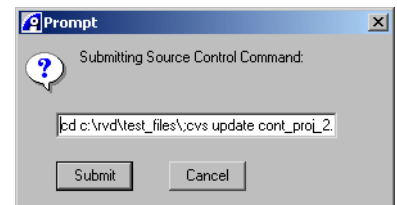
Figure 13-3 Source control command prompt

The data field shows the command ready for submission. You can edit the command in this field before submission. Click **Submit** to execute the command shown in the command field.

Click **Cancel** to abort the submission and close the prompt box.

If you submit version control commands, the **SrcCtrl** tab is brought to the front of the Output pane. This displays submitted commands and any messages returned from your version control tool.

13.3 Configuring a custom version control tool

You can specify the version control system you are using or set up RealView Debugger to handle a custom tool. If you choose to specify a custom tool, you must also define the commands to use.

To configure version control settings you must change your workspace settings. This means that the settings are re-used when RealView Debugger next starts up with the current workspace. This section describes how to do this:

- *Changing your workspace settings*
- *Specifying custom commands* on page 13-10.

———— **Note** ————

If you are working without a workspace, select **Tools → Options...** to display the Options window to make the changes described in the rest of this chapter. Changing version control rules in your global configuration file means that they apply across all workspaces and are available when you are working without a workspace.

13.3.1 Changing your workspace settings

To examine, or change, your current workspace settings, select **Tools → Workspace Options...**. This displays the Workspace Options window, described in detail in the chapter describing configuring workspaces in *RealView Developer Kit v2.2 Debugger User Guide*.

To configure version control settings for the current workspace:

1. Expand the hierarchy to display the Src_ctrl settings in the rvdebug.ini file, shown in Figure 13-4.
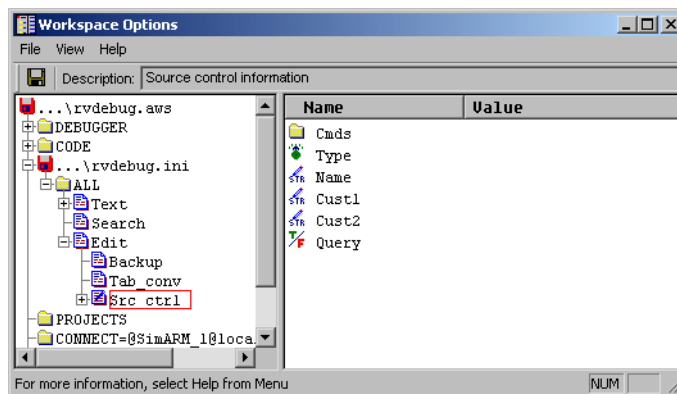
Figure 13-4 Source control settings in the workspace

2. Change the values as required, for example select the toggle box to set Query to True. Changing this setting displays the command prompt for confirmation before submitting a command.

3. Select **File → Save and Close** to save the new settings and close the Workspace Options window.

Table 13-1 gives a summary of source control settings that can be configured in the workspace settings file.

Table 13-1 Src_ctrl settings in workspace

Name	Properties
Type	Type of version control system. Right-click to see a list of suggested entries.
Name	Name of version control tool.
Cust1	The menu name for the first custom command. For example, this replaces the option **Show Tags** in the **Source control** menu, shown in Figure 13-2 on page 13-5.
Cust2	The menu name for the second custom command. For example, this replaces the option **File Status** in the **Source control** menu, shown in Figure 13-2 on page 13-5.
Query	Specifies if a command prompt is displayed for confirmation before submission. This is set to False by default.

13.3.2 Specifying custom commands

If you specify a custom control tool, you must also specify the valid source control commands:

1. Expand the hierarchy to display the source control settings `Cmds` container in the `rvdebug.ini` file, shown in Figure 13-5.
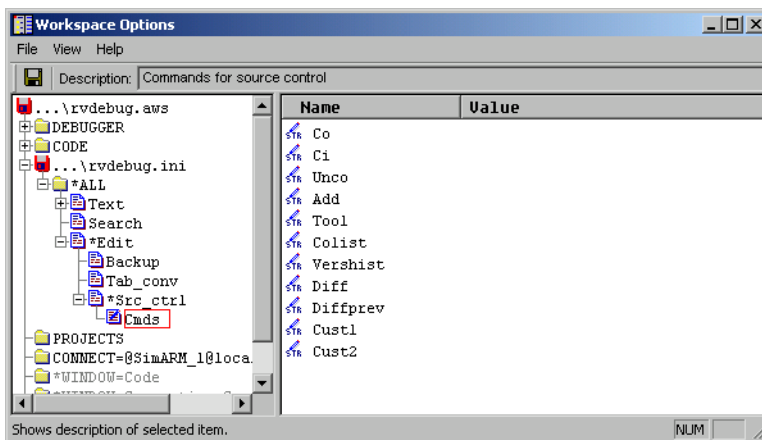
Figure 13-5 Custom source control commands in the workspace

2. Change the values as required, for example right-click on the `Co` string entry and select **Edit Value** from the context menu. Enter the command used to check files out for editing as defined by your custom control tool.

3. Select **File → Save and Close** to save the new settings and close the Workspace Options window.

——— **Note** ———

You can also use these settings to override known commands, for example to specify flags to a command or to replace a command with an alias.

———————————————

                   ARM DUI 0291C

Table 13-2 gives a summary of custom command settings that can be configured in the workspace settings file.

Table 13-2 Custom source control commands in workspace

Name	Properties
Co	Command to check files out for editing
Ci	Command to check files in for editing
Unco	Command to cancel checkout
Add	Command to add a file to source control
Tool	Source control tool to use
Colist	Command to show checkout list
Vershist	Command to show version history
Diff	Command to show differences between two versions
Diffprev	Command to show differences between current and previous versions
Cust1	First custom command, submitted when the first custom command is selected from the **Source Control** menu
Cust2	Second custom command, submitted when the second custom command is selected from the **Source Control** menu

———— **Note** ————

Changes made to these workspace settings take immediate effect. However, setting custom menu names to empty does not take effect until RealView Debugger is restarted because this is the point when defaults are restored.

————————————

Glossary

The items in this glossary are listed in alphabetical order, with any symbols and numerics appearing at the end.

Access-provider connection

A debug target connection item that can connect to one or more target processors. The term is normally used when describing the RealView Debugger Connection Control window.

Address breakpoint   A type of breakpoint.

See also Breakpoint.

ADS   *See* ARM Developer Suite.

ARM Developer Suite (ADS)

A suite of software development applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of *RISC* processors. ADS is superseded by RealView Developer Suite (RVDS).

See also RealView Developer Suite.

ARM instruction   A word that encodes an operation for an ARM processor operating in ARM state. ARM instructions must be word-aligned.

Copyright © 2005, 2006 ARM Limited. All rights reserved.

ARM state
A processor that is executing ARM instructions is operating in ARM state. The processor switches to Thumb state (and to recognizing Thumb instructions) when directed to do so by a state-changing instruction such as BX, BLX.

See also Thumb state.

Asynchronous execution
Asynchronous execution of a command means that the debugger accepts new commands as soon as this command has been started, enabling you to continue do other work with the debugger.

Backtracing
See Call Stack.

Big-endian
Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word.

See also Little-endian.

Board
RealView Debugger uses the term *board* to refer to a target processor, memory, peripherals, and debugger connection method.

Board file
The *board file* is the top-level configuration file, normally called rvdebug.brd, that references one or more other files.

Breakpoint
A user defined point at which execution stops in order that a debugger can examine the state of memory and registers.

See also Hardware breakpoint and Software breakpoint.

Call Stack
This is a list of procedure or function call instances on the current program stack. It might also include information about call parameters and local variables for each instance.

Conditional breakpoint
A breakpoint that halts execution when a particular condition becomes True. The condition normally references the values of program variables that are in scope at the breakpoint location.

Context menu
See Pop-up menu.

Core module
In the context of Integrator, an add-on development board that contains an ARM processor and local memory. Core modules can run stand-alone, or can be stacked onto Integrator motherboards.

See also Integrator.

Current Program Status Register (CPSR)
See Program Status Register.

DCC *See* Debug Communications Channel.

Debug Communications Channel (DCC)

A debug communications channel enables data to be passed between RealView Debugger and the EmbeddedICE logic on the target using the JTAG interface, without stopping the program flow or entering debug state.

Debug With Arbitrary Record Format (DWARF)

ARM code generation tools generate debug information in DWARF2 format by default. From RVCT v2.2, you can optionally generate DWARF3 format (Draft Standard 9).

Deprecated A deprecated option or feature is one that you are strongly discouraged from using. Deprecated options and features are to be removed in future versions of the product.

Doubleword A 64-bit unit of information.

DWARF *See* Debug With Arbitrary Record Format.

ELF *See* Executable and Linking Format.

Embedded Trace Macrocell (ETM)

A block of logic, embedded in the hardware, that is connected to the address, data, and status signals of the processor. It broadcasts branch addresses, and data and status information in a compressed protocol through the trace port. It contains the resources used to trigger and filter the trace output.

EmbeddedICE logic The EmbeddedICE logic is an on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.

See also IEEE1149.1.

Emulator In the context of target connection hardware, an emulator provides an interface to the pins of a real core (emulating the pins to the external world) and enables you to control or manipulate signals on those pins.

Endpoint connection

A debug target processor, normally accessed through an *access-provider connection*.

ETM *See* Embedded Trace Macrocell.

ETV *See* Extended Target Visibility.

Executable and Linking Format (ELF)

The industry standard binary file format used by RealView Developer Kit. 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 a partially linked ELF object.

Extended Target Visibility (ETV)

Extended Target Visibility enables RealView Debugger to access features of the underlying target, such as chip-level details provided by the hardware manufacturer or SoC designer.

Floating Point Emulator (FPE)

Software that emulates the action of a hardware unit dedicated to performing arithmetic operations on floating-point values.

FPE   *See* Floating Point Emulator.

Halfword   A 16-bit unit of information.

Hardware breakpoint

A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in *Read Only Memory* (ROM). Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system.

See also Breakpoint and Software breakpoint.

IEEE 1149.1   The IEEE Standard that defines TAP. Commonly (but incorrectly) referred to as JTAG.

See also Test Access Port

Integrator   A range of ARM hardware development platforms. *Core modules* are available that contain the processor and local memory.

Joint Test Action Group (JTAG)

An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a *Joint Test Action Group* (JTAG) interface port to communicate with processors. For more information see IEEE Standard, Test Access Port and Boundary-Scan Architecture specification 1149.1 (JTAG).

JTAG   *See* Joint Test Action Group.

JTAG interface unit   A protocol converter that converts low-level commands from RealView Debugger into JTAG signals to the processor, for example to the EmbeddedICE logic and the ETM.

Little-endian   Memory organization where the least significant byte of a word is at the lowest address and the most significant byte is at the highest address of the word.

See also Big-endian.

Pop-up menu   Also known as *Context menu*. A menu that is displayed temporarily, offering items relevant to your current situation. Obtainable in most RealView Debugger windows or panes by right-clicking with the mouse pointer inside the window. In some windows the pop-up menu can vary according to the line the mouse pointer is on and the tabbed page that is currently selected.

Processor core   The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.

Profiling   Accumulation of statistics during execution of a program being debugged, to measure performance or to determine critical areas of code.

Program Status Register (PSR)

Contains information about the current execution context. It is also referred to as the *Current PSR* (CPSR), to emphasize the distinction between it and the *Saved PSR* (SPSR), which records information about an alternate processor mode.

PSR   *See* Program Status Register.

RealView Compilation Tools (RVCT)

RealView Compilation Tools is a suite of tools, together with supporting documentation and examples, that enables you to write and build applications for the ARM family of *RISC* processors.

RealView Connection Broker

RealView Connection Broker is an execution vehicle that enables you to connect to simulator targets on your local system, or on a remote system.

RealView Developer Kit (RVDK)

RealView Developer Kit is a suite of software development applications, together with supporting documentation and examples, that enables you to write and debug applications for the ARM family of *RISC* processors.

RealView ICE (RVI)   A JTAG-based debug solution to debug software running on ARM processors.

RealView ICE Micro Edition (RVI-ME)

A JTAG-based debug tool for embedded systems.

RealView Trace   Provides tracing functionality for RealView ICE.

RTOS   Real Time Operating System.

RVCT   *See* RealView Compilation Tools.

RVDK   *See* RealView Developer Kit.

RVI   *See* RealView ICE.

RVI-ME   *See* RealView ICE Micro Edition.

RVT   *See* RealView Trace.

Scan chain	A scan chain is made up of serially-connected devices that implement boundary-scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain. Processors might contain several shift registers to enable you to access selected parts of the device.
Scope	The range within which it is valid to access such items as a variable or a function.
Script	A file specifying a sequence of debugger commands that you can submit to the command-line interface using the include command.
Semihosting	A mechanism whereby I/O requests made in the application code are communicated to the host system, rather than being executed on the target.
Simulator	A simulator executes non-native instructions in software (simulating a core).
Software breakpoint	A *breakpoint* that is implemented by replacing an instruction in memory with one that causes the processor to take exceptional action. Because instruction memory must be altered software breakpoints cannot be used where instructions are stored in read-only memory. Using software breakpoints can enable interrupt processing to continue during the breakpoint, making them more suitable for use in real-time systems.
	See also Breakpoint and Hardware breakpoint.
Software Interrupt (SWI)	
	An instruction that causes the processor to call a programmer-specified subroutine. Used by the ARM standard C library to handle semihosting.
SPSR	Saved Program Status Register.
	See also Program Status Register.
SWI	*See* Software Interrupt.
Synchronous execution	
	Synchronous execution of a command means that the debugger stops accepting new commands until this command is complete.
Synchronous starting	
	Setting several processors to a particular program location and state, and starting them together.
Synchronous stopping	
	Stopping several processors in such a way that they stop executing at the same instant.
TAP	*See* Test Access Port.

TAP Controller　　　Logic on a device which enables access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.

　　　　　　　　　　　See also Test Access Port and IEEE1149.1.

Target　　　　　　　The target hardware, including processor, memory, and peripherals, real or simulated, on which the target application is running.

Target vehicle　　　Target vehicles provide RealView Debugger with a standard interface to disparate targets so that the debugger can connect easily to new target types without having to make changes to the debugger core software.

Target Vehicle Server (TVS)
　　　　　　　　　　　Essentially the debugger itself, this contains the basic debugging functionality. TVS contains the run control, base multitasking support, much of the command handling, and target knowledge, such as memory mapping, lists, rule processing, board file and .bcd files, and data structures to track the target environment.

Test Access Port (TAP)
　　　　　　　　　　　The port used to access the TAP Controller for a given device. Comprises **TCK**, **TMS**, **TDI**, **TDO**, and **nTRST** (optional).

Thumb instruction　One halfword or two halfwords that encode an operation for an ARM processor operating in Thumb state. Thumb instructions must be halfword-aligned.

Thumb state　　　　A processor that is executing Thumb instructions is operating in Thumb state. The processor switches to ARM state (and to recognizing ARM instructions) when directed to do so by a state-changing instruction such as BX, BLX.

　　　　　　　　　　　See also ARM state.

Tracepoint　　　　A tracepoint can be a line of source code, a line of assembly code, or a memory address.

Tracing　　　　　The real-time recording of processor activity (including instructions and data accesses) that occurs during program execution.

Trigger　　　　　In the context of breakpoints, a trigger is the action of noticing that the breakpoint has been reached by the target and that any associated conditions are met.

　　　　　　　　　　　In the context of tracing, a trigger is an event that instructs the debugger to stop collecting trace and display the trace information around the trigger position, without halting the processor. The exact information that is displayed depends on the position of the trigger within the buffer.

TVS　　　　　　　*See* Target Vehicle Server.

Vector Floating Point (VFP)
　　　　　　　　　　　A standard for floating-point coprocessors where several data values can be processed by a single instruction.

VFP   *See* Vector Floating Point.

Watch   A watch is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes. The Watch pane is part of the RealView Debugger Code window that displays the watches you have defined.

Watchpoint   In RealView Debugger, this is a hardware breakpoint.

Word   A 32-bit unit of information.

                   ARM DUI 0291C

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.

ARM DUI 0291C