

ARM® Workbench IDE

Version 4.1

User Guide



ARM Workbench IDE

User Guide

Copyright © 2006-2008, 2010 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History			
Date	Issue	Confidentiality	Change
March 2006	A	Non-Confidential	Release v3.0 for ARM® RealView® Development Suite
March 2007	B	Non-Confidential	Release v3.1 for RealView Development Suite
July 2007	C	Non-Confidential	Release v3.1 for RealView Development Suite - ARM flash programmer and assembler editor update
December 2007	D	Non-Confidential	Release v3.1 for RealView Development Suite - CodeWarrior Importer update
September 2008	E	Non-Confidential	RealView Development Suite v4.0 Release
28 May 2010	F	Non-Confidential	RealView Development Suite v4.1 Release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. 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.

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

Confidentiality Status

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

Product Status

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

Web Address

<http://www.arm.com>

Contents

ARM Workbench IDE User Guide

	Preface	
	About this book	vi
	Feedback	ix
Chapter 1	Introduction	
	1.1 About the workbench	1-2
	1.2 About ARM plug-ins	1-3
Chapter 2	Getting Started	
	2.1 Launching the workbench	2-2
	2.2 Workbench features	2-5
	2.3 Editing source code	2-12
	2.4 Configuring the workbench	2-14
	2.5 Builds	2-17
	2.6 Importing and exporting	2-19
	2.7 Getting help	2-22
	2.8 Restrictions of use	2-25
Chapter 3	Working with Projects	
	3.1 About ARM project types	3-2
	3.2 Creating a new RealView project	3-5
	3.3 Importing an existing Eclipse project	3-8
	3.4 Importing an existing CodeWarrior project	3-10
	3.5 Adding a file to your project	3-13
	3.6 Adding a library to your project	3-14
Chapter 4	Configuring the Build and Compilation Tools	
	4.1 Accessing the build properties for an ARM project	4-2

	4.2	Accessing the build properties for a specific file	4-3
	4.3	Configuring the ARM Compiler toolchain	4-4
	4.4	Using the ARM fromelf utility	4-5
	4.5	Restoring defaults	4-6
Chapter 5	Working with Editors		
	5.1	C/C++ editor	5-2
	5.2	ARM assembler editor	5-3
	5.3	Properties editor	5-4
	5.4	Scatter file editor	5-12
	5.5	ELF content editor	5-15
Chapter 6	Working with the ARM Flash Programmer		
	6.1	About the ARM flash programmer	6-2
	6.2	Programming a flash device	6-4
	6.3	Importing a flash image	6-6
	6.4	Managing flash targets	6-7
	6.5	Using the flash device manager	6-9
	6.6	Creating a new flash algorithm	6-11
	6.7	Exporting a board for use with RealView Debugger	6-14
	6.8	Exporting a flash device for use with RealView Debugger	6-15
Chapter 7	Working with RealView Debugger		
	7.1	Loading your executable image into RealView Debugger	7-2
	7.2	Creating your debug configuration	7-4
	7.3	Setting up your debug configuration	7-5
	7.4	Launching RealView Debugger using your debug configuration	7-7
	7.5	Exporting IP-XACT design files for use with RealView Debugger	7-8
Appendix A	Terminology, Shortcuts and Icons		
	A.1	Terminology	A-2
	A.2	Keyboard shortcuts	A-3
	A.3	Menu and toolbar icons	A-5

Preface

This preface introduces the *ARM® Workbench IDE User Guide*. It contains the following sections:

- *About this book* on page vi
- *Feedback* on page ix.

About this book

This book introduces the ARM Workbench IDE, and describes how you can use it with other tools from the ARM RealView® Development Suite. The workbench is based on Eclipse but is solely focused on building, debugging, monitoring, and managing projects for ARM targets.

This book is not intended to familiarize you with all aspects of the workbench. For information on other features not described in this guide, use the standard *Workbench User Guide* or the *C/C++ Development User Guide* in the dynamic help. See *Dynamic help* on page 2-22 for more information.

Intended audience

This book is written for developers who are using the workbench to manage their ARM-targeted development projects under Microsoft Windows or Unix. It assumes that you are an experienced software developer, and that you are familiar with the RealView tools. It does not assume that you are familiar with the workbench.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the workbench and the ARM plug-ins provided with the workbench.

Chapter 2 *Getting Started*

Read this chapter for information on the build process, build configurations, workbench features, launching the workbench, and how to use the workspace.

Chapter 3 *Working with Projects*

Read this chapter for information about using the different ARM project types, how to create new projects, import existing projects and add files to a project.

Chapter 4 *Configuring the Build and Compilation Tools*

Read this chapter for information on configuring the compilation tools to control the way the workbench builds your projects.

Chapter 5 *Working with Editors*

Read this chapter for information on the different editors provided with the workbench.

Chapter 6 *Working with the ARM Flash Programmer*

Read this chapter for information on how to configure and use flash devices with the workbench.

Chapter 7 *Working with RealView Debugger*

Read this chapter for information on how to connect to and use RealView Debugger from within the workbench.

Appendix A *Terminology, Shortcuts and Icons*

Read this appendix for information on the workbench concepts and meanings.

This book assumes that the ARM software is installed in the default location. For example, on Windows this might be *volume:\Program Files\ARM*. This is assumed to be the location of *install_directory* when referring to path names. For example:

`install_directory\RVDS\Examples\...`

You might have to change this if you have installed your ARM software in a different location.

Typographical conventions

The following typographical conventions are used in this book:

<i>italic</i>	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>monospace</u>	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
monospace bold	Denotes language keywords when used outside example code.

Further reading

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

ARM periodically provides updates and corrections to its documentation. See <http://infocenter.arm.com/help/index.jsp> for current errata sheets, addenda, and the ARM *Frequently Asked Questions* (FAQs).

ARM publications

See the following publications for detailed documentation on various components of RealView Development Suite:

- *RealView® Development Suite Getting Started Guide* (ARM DUI 0255)
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain* (ARM DUI 0529)
- *ARM® Compiler toolchain Developing Software for ARM® Processors* (ARM DUI 0471)
- *ARM® Compiler toolchain Using the Assembler* (ARM DUI 0473)
- *ARM® Compiler toolchain Assembler Reference* (ARM DUI 0489)
- *ARM® Compiler toolchain Using the Compiler* (ARM DUI 0472)
- *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries* (ARM DUI 0483)
- *ARM® Compiler toolchain Compiler Reference* (ARM DUI 0491)
- *ARM® Compiler toolchain Using ARM® C and C++ Libraries and Floating-Point Support* (ARM DUI 0475)

- *ARM® Compiler toolchain ARM® C and C++ Libraries and Floating-Point Support Reference* (ARM DUI 0492)
- *ARM® Compiler toolchain Using the Linker* (ARM DUI 0474)
- *ARM® Compiler toolchain Linker Reference* (ARM DUI 0493)
- *ARM® Compiler toolchain Creating Static Software Libraries with armar* (ARM DUI 0476)
- *ARM® Compiler toolchain Using the fromelf Image Converter* (ARM DUI 0477)
- *ARM® Compiler toolchain Errors and Warnings Reference* (ARM DUI 0496)
- *ARM® Compiler toolchain Migration and Compatibility* (ARM DUI 0530)
- *RealView® Debugger Essentials Guide* (ARM DUI 0181)
- *RealView® Debugger User Guide* (ARM DUI 0153)
- *DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *DSTREAM System and Interface Design Reference* (ARM DUI 0499)
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *RealView® ICE and RealView Trace Setting Up the Hardware* (ARM DUI 0515)
- *RealView® ICE and RealView Trace System and Interface Design Reference* (ARM DUI 0517).

Other publications

This book provides information specific to the workbench provided by ARM. For more information on Eclipse, visit the Eclipse web site at <http://www.eclipse.org>.

Feedback

ARM Limited welcomes feedback on both the workbench, and its documentation.

Feedback on the workbench

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

- Your name and company.
- The serial number and version number of your product.
- The version numbers of the C/C++ Development Tools, and the workbench. To obtain this information, select **Help** → **About ARM Workbench IDE**, and then click on **Plug-in Details**.
- Details and version numbers for all installed components, such as the hardware platform, operating system, GNU make and JRE.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tools, including the version number and build numbers.

Feedback on this book

If you have any comments on this book, send email to errata@arm.com giving:

- the title
- the number, ARM DUI 0330F
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1

Introduction

This chapter gives an overview of the workbench, its main features and installation requirements. It includes the following:

- *About the workbench* on page 1-2
- *About ARM plug-ins* on page 1-3.

See Appendix A *Terminology, Shortcuts and Icons* for more information on the terminology, shortcuts, and icons that are used by the workbench.

1.1 About the workbench

The workbench is an *Integrated Development Environment* (IDE) that combines software development with the compilation and debug technology of the ARM RealView tools. You can use it as a project manager to create, build, debug, monitor, and manage projects for ARM targets. It uses a single folder called a workspace to store files and folders related to specific projects.

See Chapter 2 *Getting Started* for more information.

1.2 About ARM plug-ins

The workbench integrates the following ARM plug-ins:

ARM Compiler toolchain

This plug-in enables you to use the ARM Compiler toolchain from within the workbench to build projects for ARM targets. It provides comprehensive configuration panels to modify the tool settings for projects and individual files.

ARM assembler editor

This plug-in provides an editor that presents ARM assembler files with customizable formatting of code that is easy to read. It also provides an auto-complete feature on labels and other navigational aids.

Properties editor

This plug-in provides an extension to the ARM assembler and C/C++ editors. You can configure your source code to provide GUI components for modifying variables or `#defines` without editing the code directly.

Scatter file editor

This plug-in provides an editor that enables you to easily create and edit scatter-loading description files.

ELF content editor

This plug-in creates tabular forms and graphical views showing the contents of image files, object files and library files.

ARM flash programmer

This plug-in provides a new project wizard to create flash algorithms and program images to targets. It also provides related export wizards for close integration with RealView Debugger.

For more information on each of these plug-ins, see:

- Chapter 4 *Configuring the Build and Compilation Tools*
- Chapter 5 *Working with Editors*
- Chapter 6 *Working with the ARM Flash Programmer*.

You can also use the dynamic help:

1. Select **Help Contents** from the **Help** menu.
2. From the Contents frame, select **ARM Workbench IDE Dynamic Help** and then select the plug-in that you want help on.

Chapter 2

Getting Started

This chapter describes the workbench, the C/C++ perspective, and associated features. It also includes configuring the workbench, building projects and how to use the different types of help provided with the workbench.

It includes the following:

- *Launching the workbench* on page 2-2
- *Workbench features* on page 2-5
- *Editing source code* on page 2-12
- *Configuring the workbench* on page 2-14
- *Builds* on page 2-17
- *Importing and exporting* on page 2-19
- *Getting help* on page 2-22
- *Restrictions of use* on page 2-25.

2.1 Launching the workbench

When launching the workbench for the first time, accept the default workspace and click **OK** to open the initial Welcome view as shown in Figure 2-1. You can choose a different workspace location later if you prefer, see *Workspace* on page 2-5 for more information.

From here you can access the following topics:

Overview Introduction to the workbench and related tools.

What's New Website links for update information on new features.

Tutorials Step by step guide through specific topics.

Workbench Development environment where you create, manage and build projects.

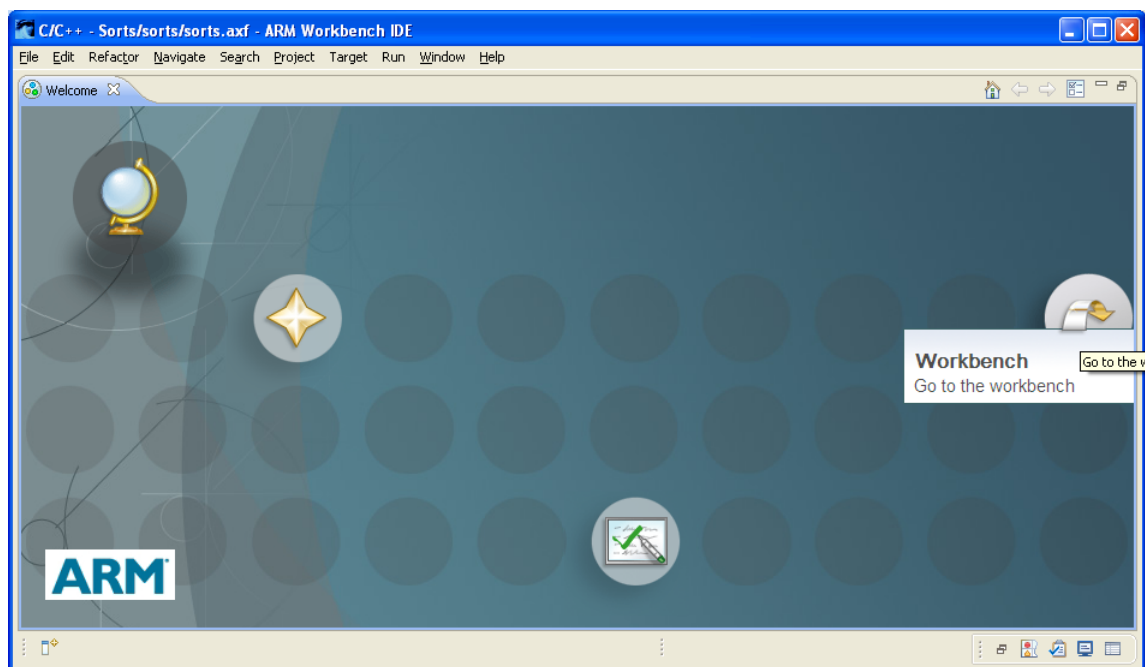


Figure 2-1 Welcome view

See also:

- *Language packs* on page 2-3
- *Opening and closing the workbench window* on page 2-3

2.1.1 Language packs

Some features of the workbench are translated into different languages. The ARM Workbench IDE provides language translations for ARM plug-ins only. You can download other language packs from the Eclipse website when available.

There are two ways to launch the workbench with a different language pack:

- If your operating system is running in the language that you want to use then the workbench automatically displays the translated features.
- If your operating system is not running in the language that you want to use then you must specify the `-nl` command-line argument when launching the workbench.

For example, to use the Japanese language pack you can use:

```
awide-4.0 -nl ja
```

The following language translations are provided with the ARM Workbench IDE:

ja	Japanese
ko	Korean
zh_CN	Simplified Chinese.

2.1.2 Opening and closing the workbench window

To open the workbench window, click on the curved arrow icon labelled **Workbench**, see Figure 2-1 on page 2-2.

Note

You can return to the Welcome view at any time by selecting **Welcome** from the **Help** menu.

Figure 2-2 on page 2-4 shows a typical workbench window displaying the C/C++ perspective and some of the associated views.

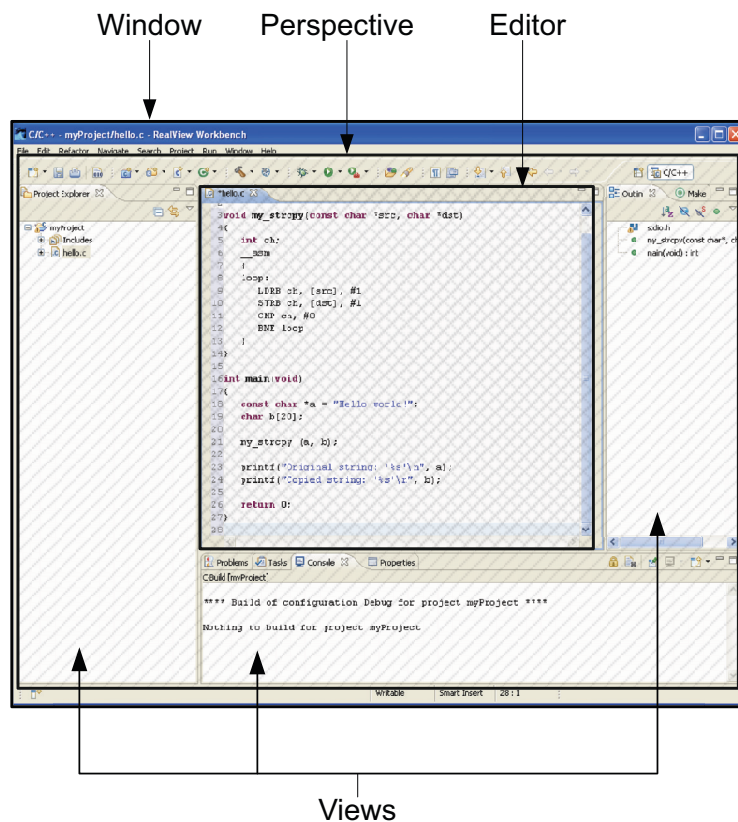


Figure 2-2 Workbench window

To close the workbench window and exit, select **Exit** from the **File** menu or click on the close icon in the top corner of the window. On exit, the workbench saves automatically so that when you next open it, the window returns to the same perspectives and views using the saved settings.

2.2 Workbench features

The workbench window is the main development environment where you can manage individual projects, associated sub-folders, and source files. Each workbench window is linked to one workspace. If you want to use different workspaces at the same time, you can launch several workbench windows and link each one to a different workspace.

The main workbench features are:

Editors Editors are special types of views used to display related source files. The tabs in the editor area show files that are currently open for editing.

Menus and Toolbars

The main menu and toolbar are located at the top of the workbench window. Other toolbars associated with specific features are located at the top of each perspective or view.

Perspectives Perspectives define the layout of your selected views and editors in the workbench. They also have their own associated menus and toolbars.

Resources Resources are projects, files, and folders that exist in your workbench.

Views Views provide related information in association with the active file in the editor. They also have their own associated menus and toolbars.

Workspace Workspace is an area designated on your file system to store files and folders related to your workbench projects and also your personal workbench settings.

See also:

- *Workspace*
- *Resources* on page 2-6
- *Perspectives and views* on page 2-9
- *Menus* on page 2-10
- *Toolbars* on page 2-11
- Chapter 5 *Working with Editors*.

2.2.1 Workspace

The workspace is an area designated on your file system to store files and folders related to your workbench projects and also your personal workbench settings.

———— **Note** ————

It is recommended that you select a dedicated workspace folder for your workbench projects only. If you select an existing folder containing non-related resources, you cannot access them in the workbench. These resources might also cause a conflict later when you create and build projects.

Changes to the customized settings in the Preferences dialog box are saved in your workspace. If you select a different workspace then these settings might be different.

When the workbench launches for the first time, the Workspace Launcher dialog box opens enabling you to select your workspace, see Figure 2-3 on page 2-6.

On subsequent launches the last saved workspace is shown as the default selection in the drop-down box. You can select another workspace by clicking on the down arrow or the **Browse...** button.

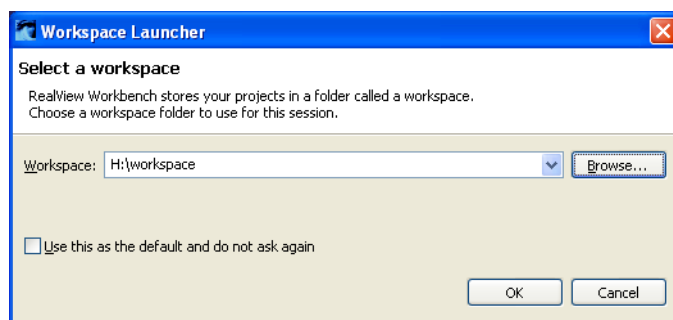


Figure 2-3 Workspace launcher dialog box

If required, select the checkbox to disable the Workspace Launcher dialog box from opening on subsequent launches.

———— **Note** ————

You can change the default workspace at any time by selecting **Switch Workspace...** from the **File** menu.

Alternatively, to open the workbench and automatically link to a specific workspace you can use the `-data` command-line argument. For example:

```
awide-4.0 -data h:\workspace
```

Editing files outside the workbench

Project sub-folders and files can be edited even when the workbench is not running. When you next launch the workbench, the default preferences enable the relevant views to refresh and update. Alternatively if you change the default preferences, you can click on the updated sub-folder or file in the Project Explorer view and select **Refresh** from the File menu.

2.2.2 Resources

A resource is a generic term used to describe a project, file, folder or a combination of these. Resources exist in the workbench but might not always exist in the workspace. There are three types of resources:

- | | |
|----------------|---|
| Project | A project is displayed in the Project Explorer view and can be stored within the workspace folder or can be a linked resource. See <i>Linked resources</i> on page 2-7 for more information.

A project must exist in the workbench before other resources can be imported or linked to that project. The project creation process creates additional configuration files and folders, for example, build properties. These additional files and folders must not be edited or deleted. |
| Folder | A folder is displayed in the Project Explorer view and can be located within the project folder or can be a linked resource. See <i>Linked resources</i> on page 2-7 for more information. |
| File | A file is displayed in the Project Explorer view and can be located within the project folder or can be a linked resource. See <i>Linked resources</i> on page 2-7 for more information. |

Linked resources

Resources can be shared between projects or they can exist in the file system outside of your selected workspace. To do this a link must be created within the workbench.

———— **Note** ————

A linked file or folder must have a project as its parent resource.

Deleting, moving or copying a linked resource only affects the link in your workbench and not the resource that it links to. However, deleting a child resource from within a linked folder also deletes it from the file system!

Linked file

To link an existing file to a project in your workspace instead of copying it, you can use the advanced settings of the New File wizard. By default the advanced options shown in Figure 2-4 are not visible, click on the <<**Advanced** button to reveal them. A path variable can also be used to reference a file. See the dynamic help for more information on using variables.

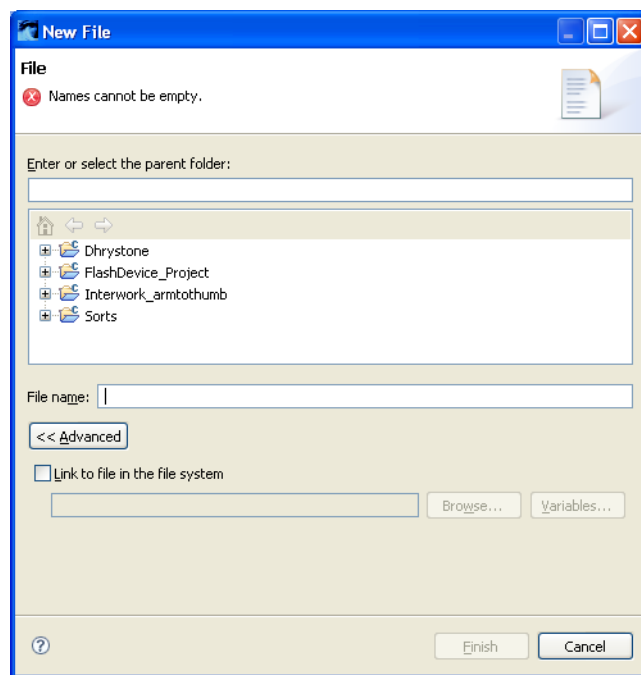


Figure 2-4 Linked file

For more information on creating a new file, see *Adding a file to your project* on page 3-13.

Linked folder

To link an existing folder to a project in your workspace instead of copying it, you can use the advanced settings of the New Folder wizard. By default the advanced options shown in Figure 2-5 on page 2-8 are not visible, click on the <<**Advanced** button to reveal them. A path variable can also be used to reference a file, see the dynamic help for more information.

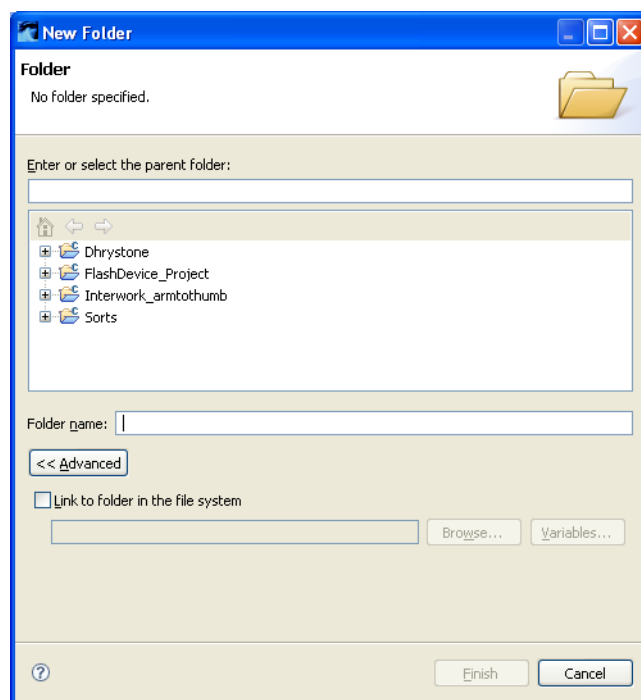


Figure 2-5 Linked folder

Linked project

The workbench uses the Import wizard to create a link to an existing project. This can be useful if you have a central folder with shared projects. By default, the **Copy projects into workspace** checkbox shown in Figure 2-6 is selected to ensure that a copy of your project is placed in your current workspace. If you want to leave your project where it is and create a link to it, you must deselect this option.

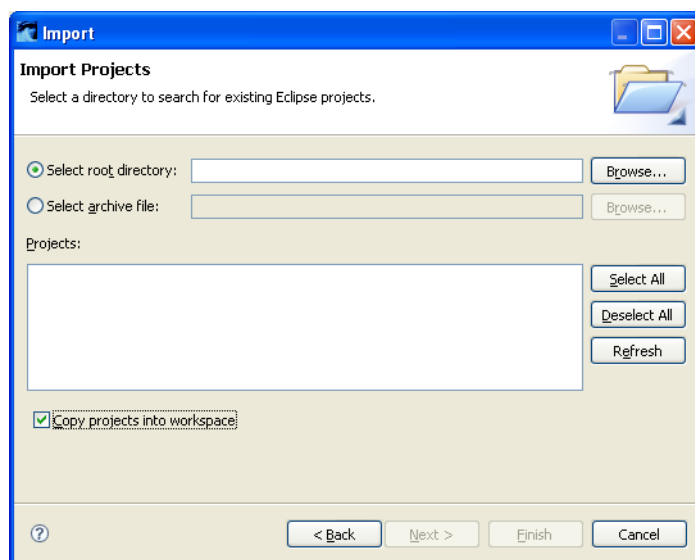


Figure 2-6 Linked project

For more information on importing existing projects, see Chapter 3 *Working with Projects*.

Disabling the use of linked resources

You can disable the use of linked resources by changing the **General** → **Workspace** settings in the Preferences dialog box. See the *Preferences dialog box* on page 2-14 for more information.

2.2.3 Perspectives and views

The main workbench window contains one or more perspectives and each perspective contains one or more views.

Perspectives

For all ARM projects you can use the C/C++ perspective.

For other projects you can change perspective by using the perspective toolbar as shown in Figure 2-7 or select **Open perspective** from the **Window** menu.

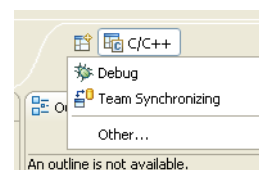


Figure 2-7 Changing perspective

Views

A view is a small visual component within the workbench to navigate through resources or to display properties for building or editing. An editor is a special type of view that enables viewing and editing of source files. The following views are associated with the workbench:

Project Explorer

This view is similar to the Navigator view and the C/C++ Projects view providing a hierarchical view of resources relevant to C/C++ project files. Right-clicking on a resource produces a context menu for specific tasks.

Console

This view provides the I/O interface for your program enabling keyboard input and textual output.

Editor

Editors are associated with specific file types and open the related editor view automatically when you open an editable file from the Project Explorer view.

There are several editors provided with the workbench:

- Standard C/C++ editor
- ARM assembler editor
- Properties editor
- Scatter file editor
- ELF content editor.

See Chapter 5 *Working with Editors* for more information.

Help

This view displays dynamic help for the selected feature when you select the question mark icon. See *Dynamic help* on page 2-22 for more information.

Outline

This view displays a structured list of C/C++ elements in the active file. Clicking on an element changes the editor focus to the position of that element in the active file.

Problems	This view displays error messages encountered during a build. Selecting an error message opens the associated file and moves the focus to the line causing the problem.
Properties	This view displays names and values for the selected item. For example, the last modified time/date for a file.

For more information on the other views not listed here, see the dynamic help.

2.2.4 Menus

The main menu is located at the top of the workbench window and can be customized to your personal preferences. The contents might vary depending on the installed plug-ins and also the active perspective.

The workbench supports the following options from the main menu:

File	This enables you to create, save, close, print, import, and export resources. You can also manage project and file property settings.
Edit	This enables you to cut, copy, paste, find, and replace text within a resource.
Refactor	This enables you to rename a selected object in your code and propagate the change through other files in the project.
Navigate	This enables you to navigate and quickly find specific resources.
Search	This provides an advanced filter for searching through resources.
Project	This enables you to manage project build configurations and perform specific builds. You can also customize the tool settings for ARM Compiler toolchain.
Target	This enables you to manage flash devices and flash targets.
Run	This enables you to run, send to, debug or configure external tools. You can also manage breakpoints and watchpoints.
Window	This enables you to open, close, and customize perspectives, views and editors.
Help	This provides documentation on the workbench and the tools provided with RealView Development Suite. You can also access the ARM cheat sheets and software updates.

Right-clicking on a resource produces a context menu for specific tasks. For more information on the other menu options not listed here, see the dynamic help.

2.2.5 Toolbars

The main toolbar is located at the top of the workbench window and can be customized to your personal preferences. Figure 2-8 shows the main workbench toolbar for the C/C++ perspective. The contents might vary depending on the installed plug-ins and also the active perspective.

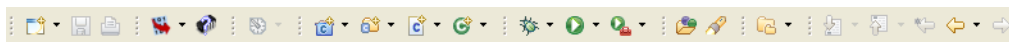


Figure 2-8 Workbench toolbar

Other toolbars associated with specific features are located at the top of each perspective or view, see Figure 2-9 and Figure 2-10.

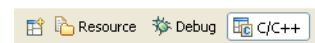


Figure 2-9 Perspective toolbar



Figure 2-10 View toolbar

See *Menu and toolbar icons* on page A-5 for more information.

2.3 Editing source code

You can use the editors provided with the workbench to edit your source code or you can use an external editor. If you work with an external editor you must refresh the workbench to synchronize the views with the latest updates. To do this, in the Project Explorer view, you can click on the updated project, sub-folder, or file and select **Refresh** from the **File** menu. Alternatively you can enable the automatic refresh option by selecting **General** → **Workspace** → **Refresh automatically** in the Preferences dialog box.

When you open a file in the workbench, a new editor tab appears with the name of the file. An edited file displays an asterisk (*) in the tab name to show that it has unsaved changes.

When you have two or more editor tabs open, you can tile them for side-by-side viewing by clicking on a tab and dragging it over an editor border.

In the left-hand margin of each editor tab you can find a vertical bar that displays markers relating to the active file.

See also:

- *Navigating*
- *Searching*
- *Content assist*
- *Bookmarks* on page 2-13
- *Preferences dialog box* on page 2-14
- *Editor markers* on page A-6.

2.3.1 Navigating

There are several ways to navigate to a specific resource within the workbench. You can use the Project Explorer view to open a resource by browsing through the resource tree and double-clicking on a file. An alternative is to use the keyboard shortcuts or use the options from the Navigate menu.

The Navigate menu enables you to locate a resource by pattern matching using the Go To option or you can use the Open Resource option to directly open a file in an editor.

2.3.2 Searching

To locate information or specific code contained within one or more files in the workbench, you can use the options from the Search menu. Textual searching with pattern matching and filters to refine the search fields are provided in a customizable Search dialog box. You can also open this dialog box from the main workbench toolbar.

2.3.3 Content assist

The C/C++ and ARM Assembler editors provide content assistance with functions and labels at the cursor position to auto-complete the selected item. Using the Ctrl+Space keyboard shortcut produces a small dialog box with a list of valid options to choose from. You can shorten the list by partially typing a few characters before using the keyboard shortcut. From the list you can use the Arrow Keys to select the required item and then press the Enter key to insert it into your code.

2.3.4 Bookmarks

Bookmarks can be used to mark a specific position in a file or mark an entire file so that you can return to it quickly. To create a bookmark, select a file or line of code that you want to mark and select **Add Bookmark** from the **Edit** menu. The Bookmarks view displays all the user defined bookmarks and can be accessed by selecting **Show View → Bookmarks** from the Window menu. If the Bookmarks view is not listed then select **Others...** for an extended list.

To delete a bookmark, open the Bookmarks view, click on the bookmark that you want to delete and select **Delete** from the Edit menu.

2.4 Configuring the workbench

The workbench can be customized to your own settings by changing the layout, key bindings, file associations, and color schemes.

Projects and files can also be configured to use the build system in different ways by modifying the properties for the selected resource. See for more information.

Perspectives can be opened and customized using the options from the Window menu or you can use the perspectives menu and toolbar. By default a perspective opens in the same window, however, you can change your default preferences to open in a new window if you prefer.

Views can be moved or docked as applicable by dragging and dropping them into position. Double-clicking on the title bar of a view toggles the maximize/minimize options or you can use the view menu and toolbar.

See also:

- *Preferences dialog box*
- *Properties dialog box* on page 2-15.

2.4.1 Preferences dialog box

Workbench settings can be customized using the Preferences dialog box, see Figure 2-11.

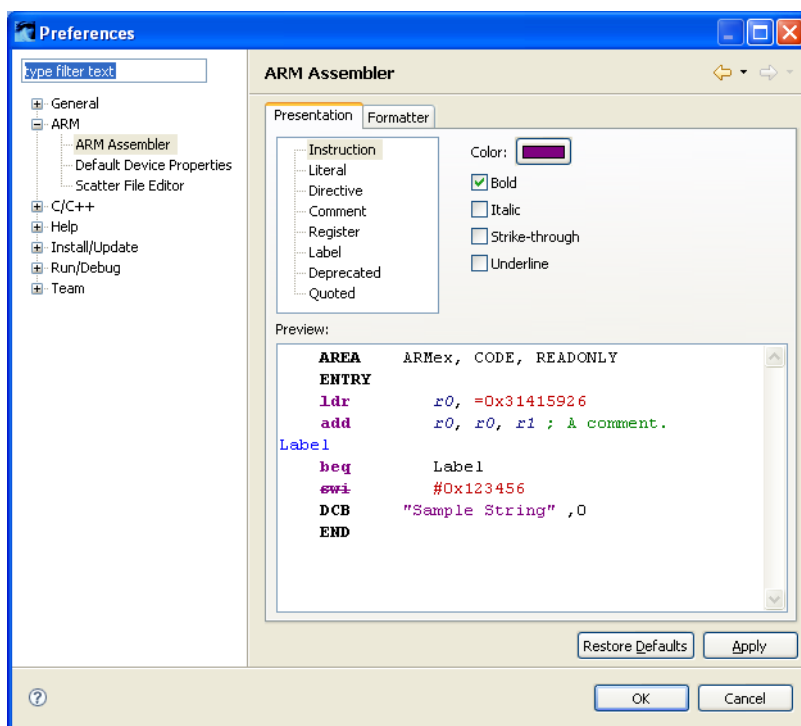


Figure 2-11 Window preferences dialog box

You can access this dialog box by selecting **Preferences...** from the **Window** menu. Changes to these settings are saved in the current workspace. If you want to copy your workbench settings to another workspace, use the Export wizard see *Importing and exporting* on page 2-19.

The contents of the preferences hierarchy tree include the following:

General Controls the workspace, build order, linked resources, file associations, path variables, background operations, keyboard and mouse settings.

ARM	Controls the presentation and formatting for ARM-specific editors and default device properties.
C/C++	Controls the C/C++ environment settings, CDT build variables, syntax formatting, and default project wizard settings.
Help	Controls how the context help is displayed.
Install/Update	Controls the update history, scheduler, and policy.
Run/Debug	Controls the default perspectives, breakpoint, build, and launch settings before running and debugging.
Team	Controls CVS synchronization.

2.4.2 Properties dialog box

Project settings can be customized using the Properties dialog box, see Figure 2-12.

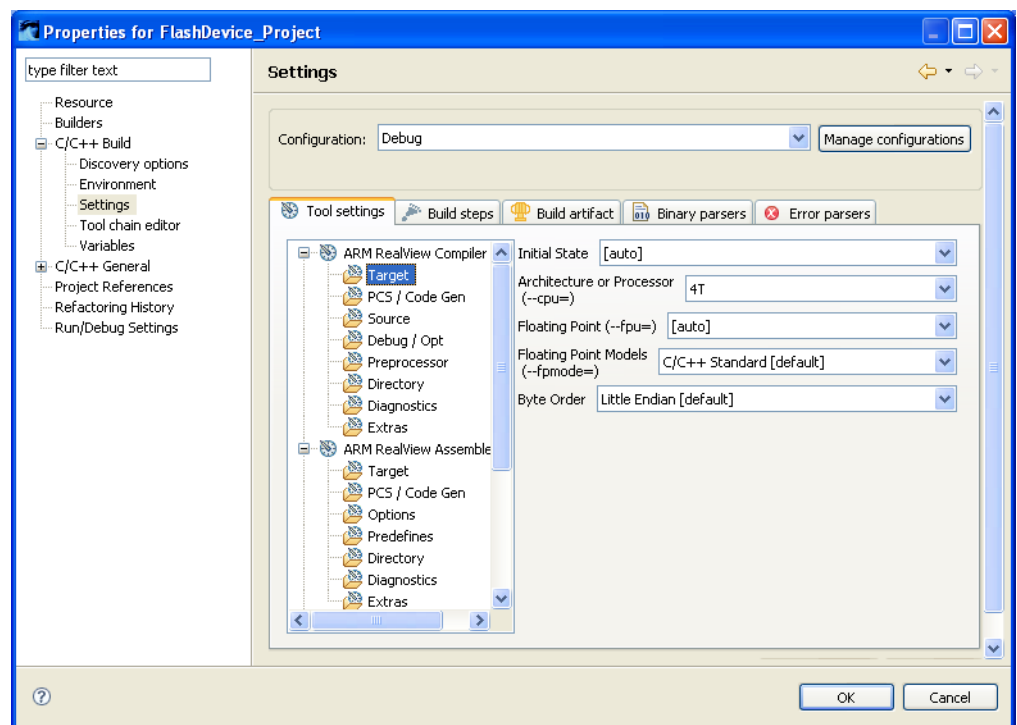


Figure 2-12 Project properties dialog box

You can access this dialog box by selecting a project and then selecting **Properties...** from the **Project** menu. Changes to the customized settings are saved in the project folder in your workspace. You can also customize the C/C++ properties for a single file for example, if you want to apply a specific compiler option to a file during the build. See Chapter 4 *Configuring the Build and Compilation Tools* for more information.

———— Note ————

If you specify different options for a single file, it overrides the options specified in the project configuration panels that apply to all related source files.

The contents of the properties hierarchy tree for a project include the following:

Resource	Displays the resource location, modification state, and file type.
Builders	Controls builders available for the selected project.

C/C++ Build

Controls the environment, build, and tool chain settings for the active configuration.

C/C++ General

Controls documentation, file types, indexer and path/symbol settings.

Project References

Controls project dependencies.

Run/Debug Settings

Controls launch configurations for the selected resource.

2.5 Builds

A build is the process of compiling and linking source files to generate an output file. A build can be applied to either a specific set of projects or the entire workspace. It is not possible to build an individual file or sub-folder. See Chapter 4 *Configuring the Build and Compilation Tools* for more information.

The workbench provides an incremental build that applies the selected build configuration to resources that have changed since the last build. Another type of build is the Clean build that applies the selected build configuration to all resources, discarding any previous build states.

Automatic

This is an incremental build that operates over the entire workspace and can run automatically when a resource is saved. This behavior must be enabled for each project by selecting **Build on resource save (Auto build)** in the project behavior dialog box, see Figure 2-13. By default, this behavior is not selected for any project.

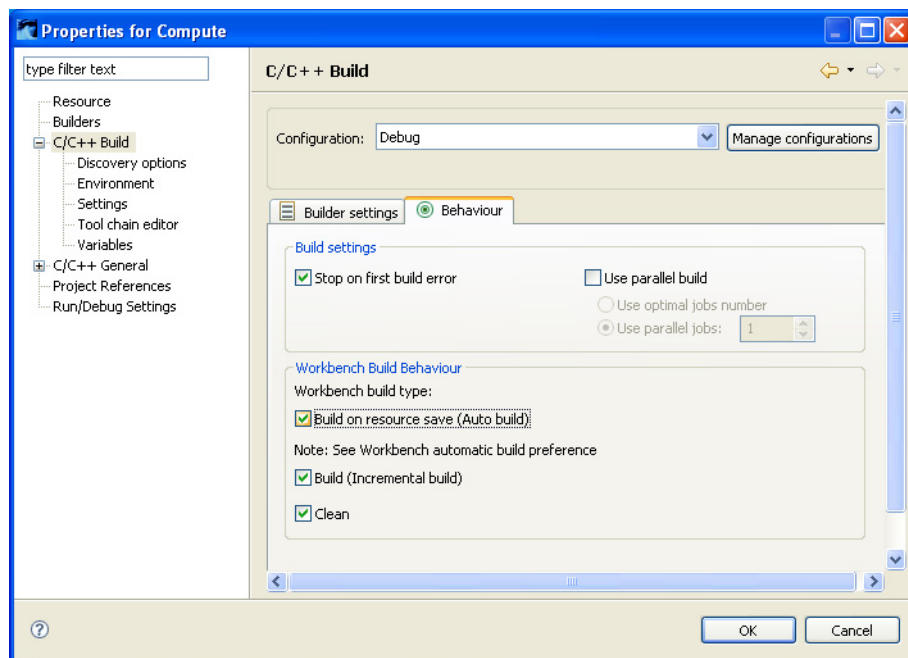


Figure 2-13 Workbench build behavior

You must also ensure that **Build Automatically** is selected from the **Project** menu. By default, this menu option is selected.

Manual

This is an incremental build that operates over the entire workspace on projects with **Build (Incremental build)** selected, see Figure 2-13. By default, this behavior is selected for all projects.

You can run an incremental build by selecting **Build All** or **Project → Build Project** from the Project menu.

Note

Manual builds do not save before running so you must save all related files before selecting this option! To save automatically before building, you can change your default settings by selecting **Preferences... → General → Workspace** from the Window menu.

Clean

This option discards any previous build states including object files and images from the selected projects. The next automatic or manual build after a clean, applies the selected build configuration to all resources.

You can run a clean build on either the entire workspace or specific projects by selecting **Clean...** from the **Project** menu. You must also ensure that **Clean** is selected in the C/C++ **Build** → **Behaviour** tab of the Preferences dialog box, see Figure 2-13 on page 2-17. By default, this behavior is selected for all projects.

Build order is a feature where inter-project dependencies are created and a specific build order is defined. For example, an image might require several object files to be built in a specific order. To do this, you must split your object files into separate smaller projects, reference them within a larger project to ensure they are built before the larger project. Build order can also be applied to the referenced projects.

For more information on build order and project references, see:

- *Preferences dialog box* on page 2-14
- *Properties dialog box* on page 2-15.

2.6 Importing and exporting

A resource must exist in a project within the workbench before you can use it in a build. If you want to use an existing resource from your file system in one of your projects, the recommended method is to use the Import wizard. To do this, select **Import...** from the **File** menu.

If you want to use a resource externally from the workbench, the recommended method is to use the Export wizard. To do this, select **Export...** from the **File** menu.

There are several options available in the import and export wizards. The workbench supports the following options:

General This option enables you to import and export the following:

- archive zip files
- files containing breakpoint settings
- complete projects
- selected source files and project sub-folders
- files containing workbench preference settings.

C/C++ This option enables you to import the following:

- C/C++ executable files
- CodeWarrior projects.

Flash programmer

This option enables you to import and export the following:

- import a flash image file into a flash project
- export board configurations for use with RealView Debugger
- export flash devices for use with RealView Debugger.

For information on the other options not listed here, see the dynamic help.

See also:

- *Importing*
- *Exporting* on page 2-20.

2.6.1 Importing

The Import wizard can be used to import complete projects, source files and, project sub-folders in addition to breakpoint and preference settings. Select **Import...** from the **File** menu.

This section focuses on projects, source files and project sub-folders.

To import a complete project either from an archive zip file or an external folder from your file system, you must use the Existing Projects into Workspace or CodeWarrior Project exported as XML wizard. This ensures that the relevant workbench files are also imported into your workspace. See Chapter 3 *Working with Projects* for more information.

Individual source files and project sub-folders can be imported using either the Archive File or File System wizard. Both options produce a dialog box similar to the example shown in Figure 2-14 on page 2-20. Using the options provided you can select the required resources and specify the relevant options, filename, and destination path.

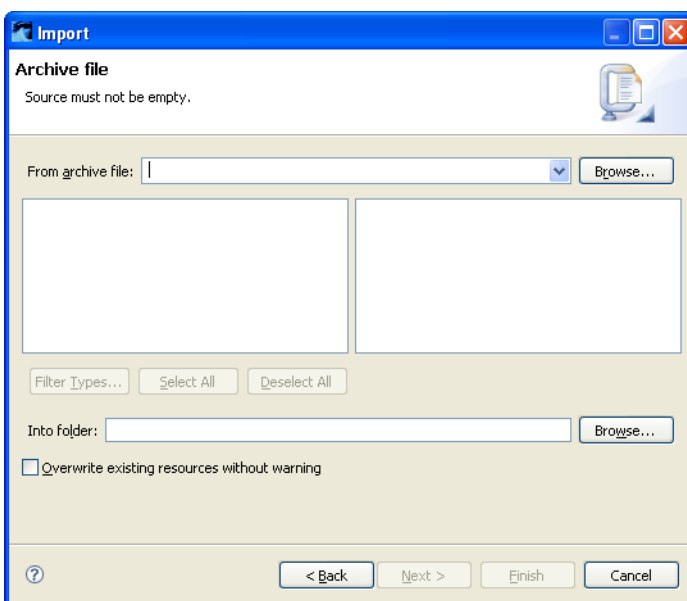


Figure 2-14 Typical example of the import wizard

With the exception of the Existing Projects into Workspace wizard, files and folders are copied into your workspace when you use the Import wizard. To create a link to an external file or project sub-folder you must use the New File or New Folder wizard, see *Linked resources* on page 2-7 for more information.

2.6.2 Exporting

The Export wizard can be used to export complete projects, source files and, project sub-folders in addition to breakpoint and preference settings. Select **Export...** from the **File** menu.

This section focuses on projects, source files and project sub-folders.

Exporting a complete project, source file or project sub-folder uses the same process. If you want to create a zip file you can use the Archive File wizard, or alternatively you can use the File System wizard. Both options produce a dialog box similar to the example shown in Figure 2-15 on page 2-21. Using the options provided you can select the required resources and specify the relevant options, filename, and destination path.

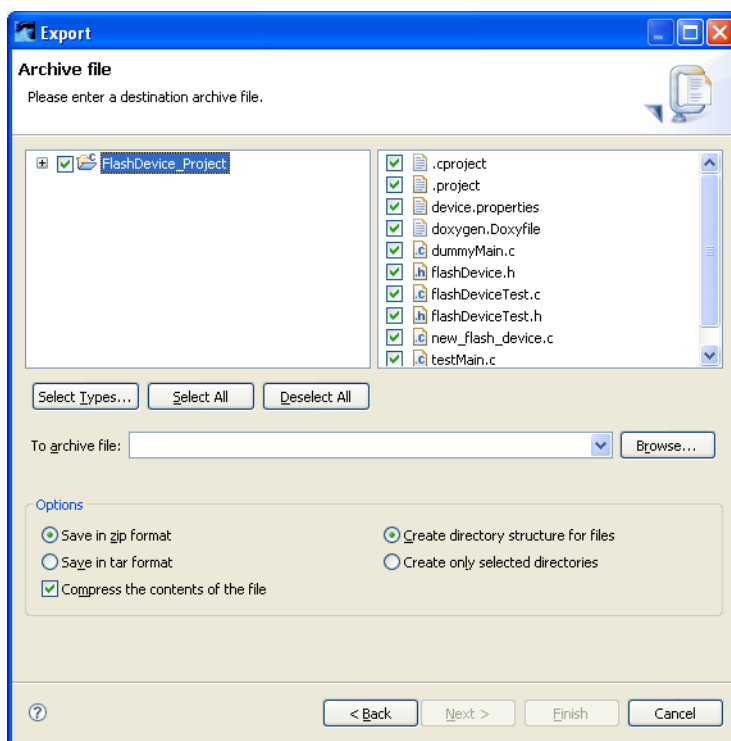


Figure 2-15 Typical example of the export wizard

2.7 Getting help

The workbench provides the following help accessible from within the workbench:

- dynamic help at the point of interest
- cheat sheets for guidance on specific tasks
- installing software updates for the workbench
- accessing the ARM website
- tutorials.

See also:

- *Dynamic help*
- *ARM specific cheat sheets*
- *Installing new features* on page 2-24
- *Accessing the ARM website* on page 2-24.

2.7.1 Dynamic help

To access the dynamic help for a specific workbench feature you must:

1. Click on an editable field for the feature that you want to use.
2. Click on the question mark icon.

Dynamic help for the selected feature appears in the About panel at the top of the Related Topics view.

Note

Other possible search results are listed in the Dynamic Help panel at the bottom of the Related Topics view.

An alternative way to view the dynamic help is to use the Help window:

1. Select **Help Contents** from the **Help** menu.
2. From the Contents frame, select **ARM Workbench IDE Dynamic Help** and then select the plug-in that you want help on.

2.7.2 ARM specific cheat sheets

Cheat sheets are working examples that you can use to guide you through a specific task. Each step in the task is listed in the Cheat Sheets view and the current step is highlighted and expanded. You must perform each step in turn to complete the task.

To use a cheat sheet:

1. Select **Cheat Sheet...** from the **Help** menu.
2. Select a cheat sheet from the list or use **Browse...** to select from a file, see Figure 2-16 on page 2-23. Click on **OK**.

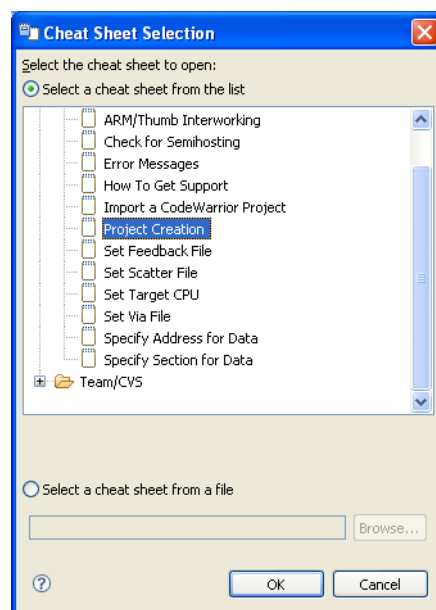


Figure 2-16 Selecting a cheat sheet

3. Click on the **Click to Begin** link on the Introduction step to start the task. If you open a cheat sheet more than once, the link on the Introduction step changes text to **Click to Restart**.
4. Follow the instructions step by step. When you complete an instruction, click on the **Click to Complete** link to move on to the next instruction. Some of the instructions might have a **Click to Perform** link for you to use if you want that instruction to be performed automatically by the cheat sheet.

The task is complete when you have performed every step listed in the Cheat Sheets view. This is shown by the tick icon shown in the left-hand margin of the cheat sheet see Figure 2-17 on page 2-24.

Figure 2-17 on page 2-24 shows a typical example of a cheat sheet. Steps one and two are shown as complete, step three (the current step) is highlighted and expanded ready for use. Four instructions are listed in step three, complete each instruction in turn. When step three is fully complete, the cheat sheet moves on to populate and reveal the instructions in step four.

Note

Subsequent steps are not populated until you fully complete the previous step.

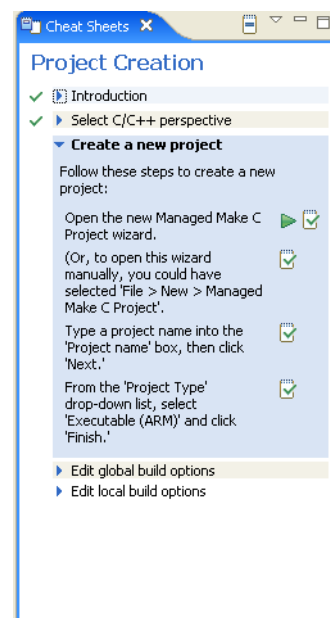


Figure 2-17 Example of a cheat sheet

2.7.3 Installing new features

To install new features:

1. Select **Help** → **Software Updates** → **Find and Install...**
2. Select **Search for new features to install** in the Install/Update dialog box. Click on **Next**.
3. Select the ARM remote site from the list of **Sites to include in search**. Click on **Finish**.
4. Read the license agreement and accept it. If you do not accept the license agreement, you cannot install that feature. Click on **Next**, and then click on **Finish**.
5. In the Verification dialog box, click on **Install All**.
6. Click on **Yes** to restart the workbench and complete the installation.

To install updated features, follow the same steps but select **Search for updates of the currently installed features** in the Install/Update dialog box.

———— **Note** ————

You can enable automatic updates in the Install/Updates panel of the Preferences dialog box. To do this, select **Preferences...** from the Window menu.

2.7.4 Accessing the ARM website

The workbench provides the following external links to the ARM website:

- ARM Documentation
- ARM Self Help Forums
- ARM Technical Support Downloads
- ARM Development Tool FAQs
- ARM Technical Support.

To access them, select **ARM on the Web** from the Help menu.

2.8 Restrictions of use

This section lists the specific restrictions and peculiarities that apply when using the workbench.

Organizing projects

The recommended structure for project source files is to create them in the project folder or sub-folder. If a source file is created in a folder that is higher than the project, an absolute link is created.

Opening an existing Eclipse project

You must use the import wizard. See *Importing an existing Eclipse project* on page 3-8 for more information.

CodeWarrior sub-projects

CodeWarrior sub-projects are not supported. You must import each project individually. See *Importing an existing CodeWarrior project* on page 3-10 for more information.

Inter-project dependencies

Nested projects are not supported. Each project must be organized as a discrete entity. Inter-project dependencies can be set up by referencing other projects that reside in your workspace. Select **Properties** → **Project References** from the Project menu to manually add references.

Link order Specifying the link order of your object files within the same project is not possible with the workbench. As a workaround, if you split your object files into different projects, you can specify the project build order. Select **Preferences...** → **General** → **Workspace** → **Build Order** from the Window menu.

Restore Defaults

Restoring the defaults of a project discards all information that is not a part of the project type. All settings changed in the ARM New Project Wizard are lost.

Starting RealView Debugger with an existing target configuration

When you launch RealView Debugger from the workbench using a project with an existing target configuration, you might see the RealView Debugger dialog box shown in Figure 2-18. Click on **No** to use the target configuration from the workbench.

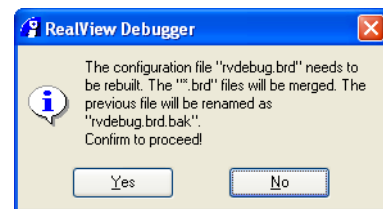


Figure 2-18 RealView Debugger message

Chapter 3

Working with Projects

You can use the workbench to create projects for ARM targets. Projects are top level folders in your workspace that contain related files and sub-folders. A project must exist in your workspace before a new file can be added or an existing file can be imported.

———— **Note** ————

The RealView Project wizard combines the features of a C Project and a C++ Project. This enables you to build .c, .cpp and .s files within the same project.

This chapter includes the following:

- *About ARM project types* on page 3-2
- *Creating a new RealView project* on page 3-5
- *Importing an existing Eclipse project* on page 3-8
- *Importing an existing CodeWarrior project* on page 3-10
- *Adding a file to your project* on page 3-13
- *Adding a library to your project* on page 3-14.

3.1 About ARM project types

This section provides information on the different ARM project types provided by the workbench. To select a project type and associated toolchain, select **File** → **New** → **RealView Project** from the main menu, see Figure 3-1.

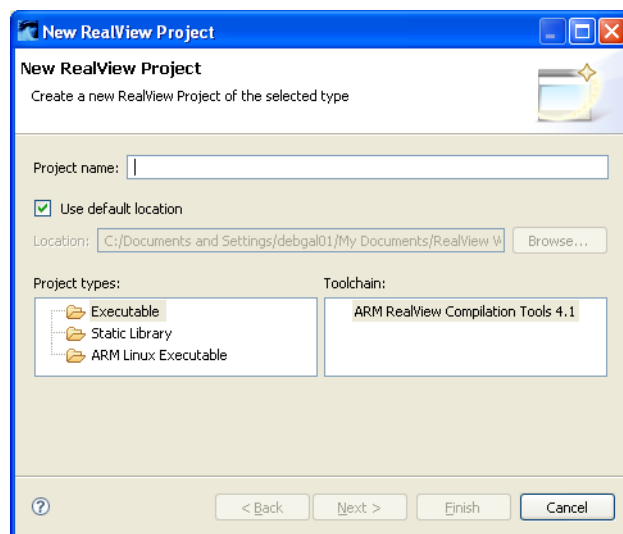


Figure 3-1 ARM project types for installed toolchains

Note

If you have several products installed, only the latest toolchain is listed in this wizard.

You can use the Project Converters dialog box to update an older project so that you can use the latest toolchain. Beware that if you revert to an earlier version you might lose toolchain functionality. To access the Project Converters dialog box, right-click on your project and select **Convert To...** from the context menu, see Figure 3-2.

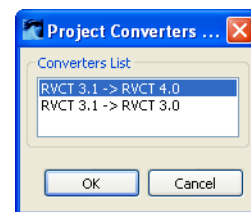


Figure 3-2 Project converters

See also:

- *Executable* on page 3-3
- *Static library* on page 3-3
- *ARM project types for Linux* on page 3-3
- *Build configurations* on page 3-3
- *Project template suite* on page 3-4.

3.1.1 Executable

Use the Executable project type to create an executable ELF image from ARM and Thumb® code. The Executable project uses:

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

You can also configure the workbench to call the ARM fromelf utility to create a binary file from the executable ELF image. See *Using the ARM fromelf utility* on page 4-5.

3.1.2 Static library

Use the Static Library project type to build a library of ELF object format members. This project type is similar to Executable. The Static Library project uses the ARM librarian (armar) to output an object library, instead of using the ARM linker (armlink) to output an executable ELF image.

Note

It is not possible to debug or run a standalone library file until it is linked into an image.

3.1.3 ARM project types for Linux

ARM projects for Linux require the CodeSourcery ARM GCC installation. See the related application note on the ARM website for more information.

Linux executable

Use the Linux Executable project type to create an executable ELF image for ARM and Thumb code. The Linux Executable project uses:

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

3.1.4 Build configurations

By default, the new project wizard provides two separate build configurations:

Debug	The debug target is configured to build output binaries that are fully debuggable, at the expense of optimization. It configures the compiler optimization setting to minimum (level 0), to provide an ideal debug view for code development.
Release	The release target is configured to build output binaries that are highly optimized, at the expense of a poorer debug view. It configures the compiler optimization setting to high (level 2). Debug information is still included in the resulting image.

In all new projects, the Debug configuration is automatically set as the active configuration. This can be changed in the C/C++ Build configuration panel. See Chapter 4 *Configuring the Build and Compilation Tools* for information.

3.1.5 Project template suite

The new project wizard provides a list of templates with default settings for specific board configurations and ARM processors. For example, ARM7TDMI® processor.

By default, the ARM7TDMI processor is selected but if you know the correct configuration of your target you can select a specific target from the list provided. Selecting the correct target ensures that all the default settings are automatically set up for you.

See *Creating a new RealView project* on page 3-5 for more information.

3.2 Creating a new RealView project

To create a new RealView project:

1. Select **File** → **New** → **RealView Project** from the main menu.
2. Enter a project name and select the type of project, see Figure 3-3. Leave the **Use Default location** option selected so that the project is created in the default folder shown. Alternatively, deselect this option and browse to your preferred project folder. Click on **Next**.

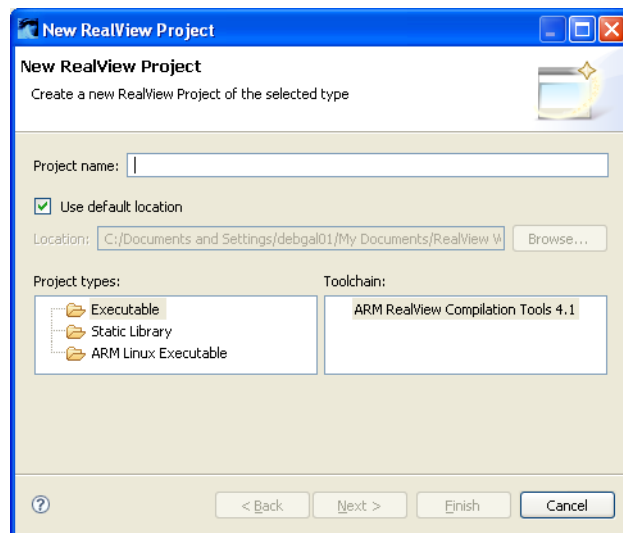


Figure 3-3 Naming your project

3. In the Configurations dialog box, leave the **Debug** and **Release** options selected to enable you to change the project settings for different configurations. Click on **Next**.
4. Figure 3-4 shows the Target dialog box with a list of templates containing default settings for specific targets. Select a template as required. Click on **Next**.

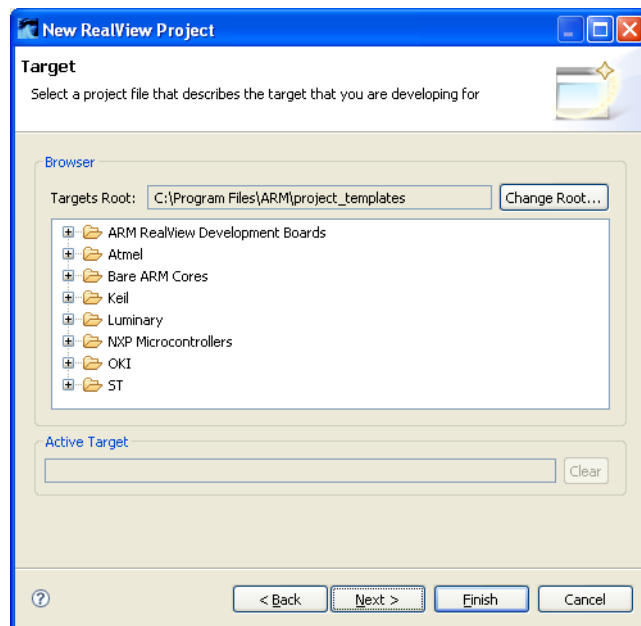


Figure 3-4 Project template selection

5. Figure 3-5 shows the Core Settings dialog box. Select **Architecture or Processor** (`--cpu=`), **Floating Point** (`--fpu=`) and **Byte Order**. Click on **Next**.

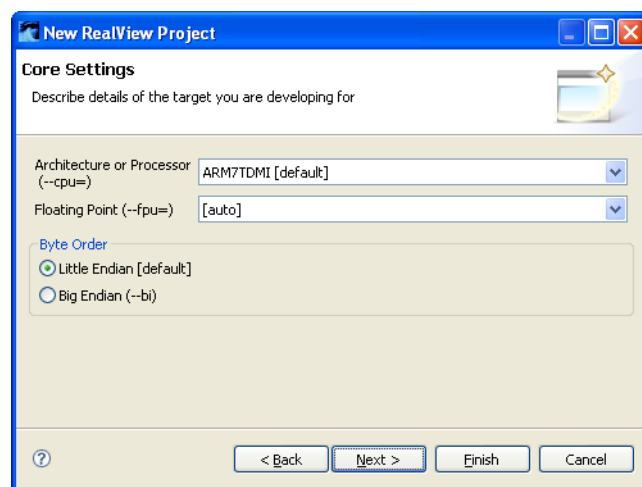


Figure 3-5 Fine-tune target settings

6. Figure 3-6 shows the Language Settings panel. Select the **Initial State** and **Source Language** if required. The default setting **[auto]** enables the compiler to automatically select the options for your project. Click on **Next**.

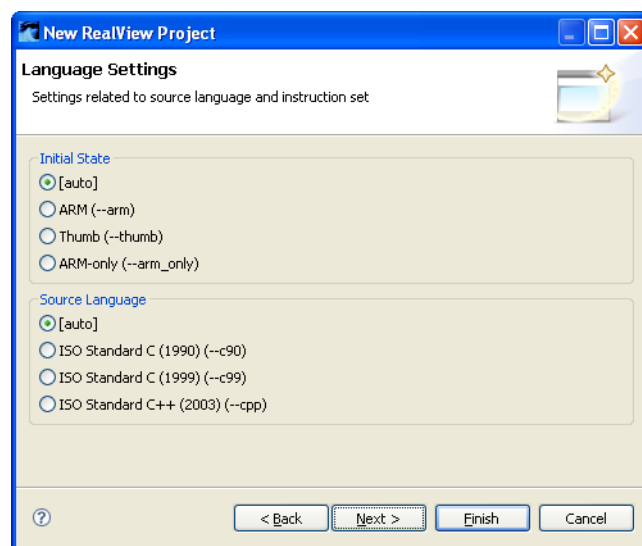
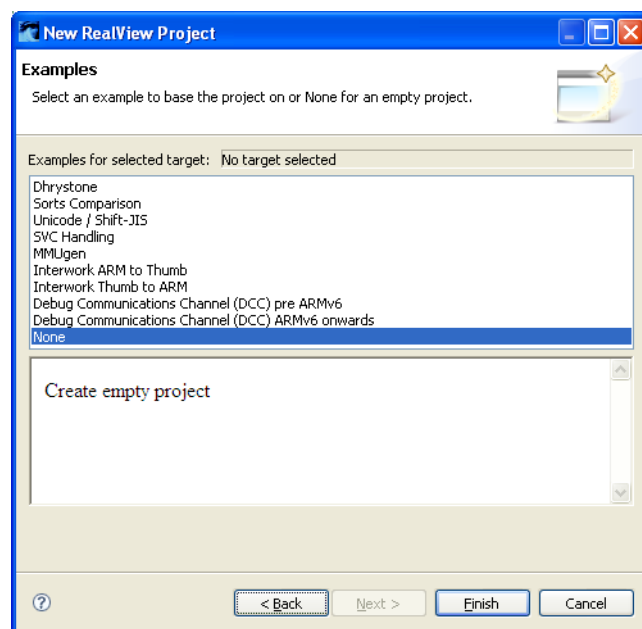


Figure 3-6 Language settings

7. For Linux projects, you are given the option to change the default configuration parameters in the Linux Target Configuration dialog box. Select as required and then click on **Finish** to create your new project. For other projects, skip this step.
8. Figure 3-7 on page 3-7 shows the Examples dialog box with a selection of examples containing code for use as a starting point for your project. The default selection, **None** creates an empty project with no example code. Select as required and then click on **Finish** to create your new project.

**Figure 3-7 Project examples**

The project is visible in the Project Explorer view.

3.3 Importing an existing Eclipse project

To import an existing Eclipse project into your workspace:

1. Select **Import...** from the **File** menu.
2. To import an existing project, select **Existing Project into Workspace** as shown in Figure 3-8. Click on **Next**.

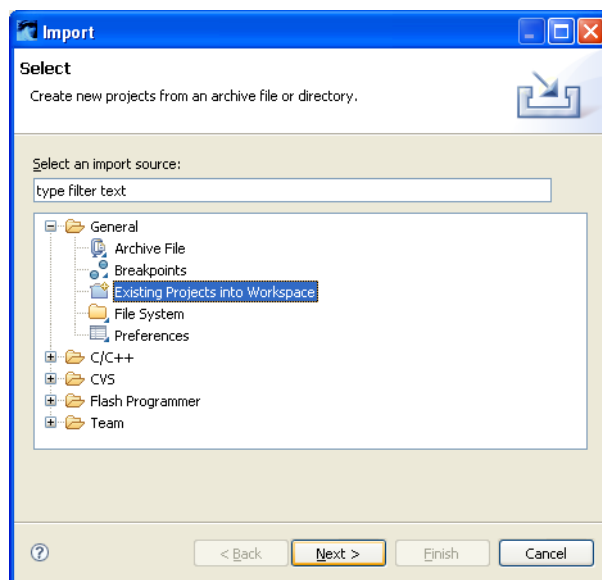


Figure 3-8 Selecting the import source type

3. Click on **Browse** to select the project root folder. Ensure that the Projects panel has all the required import project(s) selected. Select **Copy projects into workspace** if required or deselect to create links to your existing project(s) and associated files. See Figure 3-9 on page 3-9. Click on **Finish**.

———— **Note** ————

If your existing project contains project settings from an older version of the build system, you are given the option to update your project. Using the latest version means that you can access all the latest toolchain features.

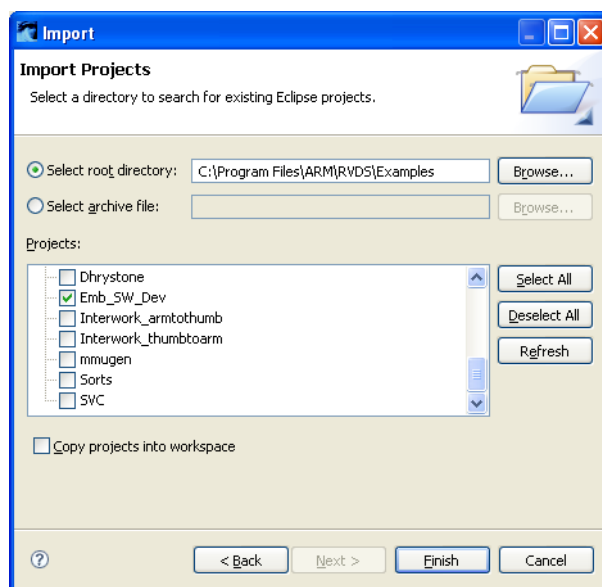


Figure 3-9 Selecting an existing Eclipse projects for import

The imported project is visible in the Project Explorer view.

3.4 Importing an existing CodeWarrior project

To import an existing CodeWarrior for RealView Development Suite project into your workspace:

1. Export your CodeWarrior project to an eXtensible Markup Language (XML) format, with a .xml filename extension.

————— Note —————

You must create the .xml file in the same folder as the CodeWarrior project file (with the .mcp filename extension)

See the *RealView® Development Suite CodeWarrior IDE Guide* for more information on exporting your CodeWarrior project into XML. When you import a CodeWarrior project into the workbench, a new project is created and the relevant tool settings are transferred and applied to the new project.

There are some limitations with the CodeWarrior importer:

- Sub-projects are not supported. You must import each project individually.
 - Sub-targets are imported in the same way as files. See *Adding a file to your project* on page 3-13 for more information.
 - Command-line options that do not map to those provided in the workbench, are included in the Extras panel for each tool. See *Configuring the ARM Compiler toolchain* on page 4-4 for more information.
2. Select **Import** from the **File** menu.
 3. Select **CodeWarrior Project exported as XML** from the Import dialog box, see Figure 3-10. Click on **Next**.

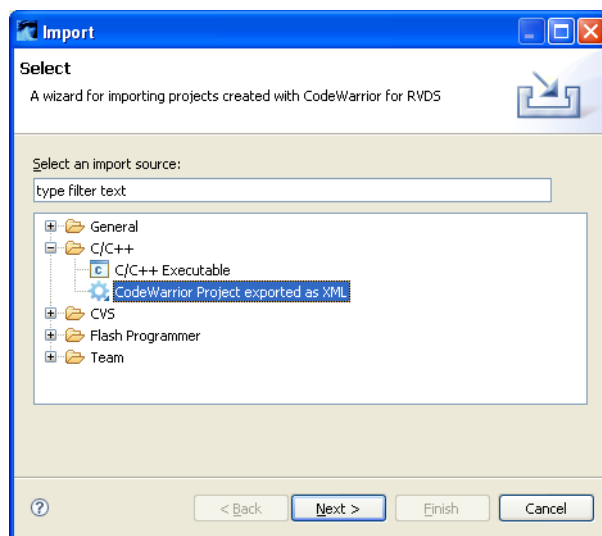


Figure 3-10 Selecting CodeWarrior XML

4. Click **Browse...** to select the folder containing the .xml file, see Figure 3-11 on page 3-11.

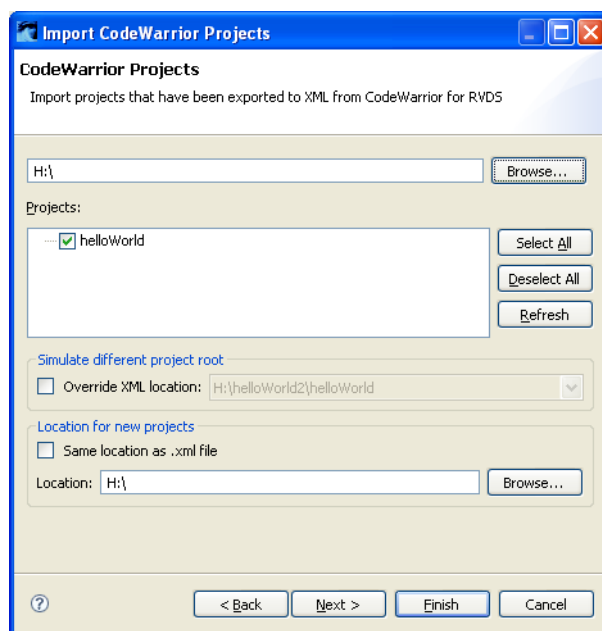


Figure 3-11 Importing a CodeWarrior Project

5. By default, all the projects that can be imported are selected. If the Projects panel shows more than one project, deselect the projects that you do not want to import.
6. If your source files are located in a folder above the CodeWarrior project root, select the **Override XML location** option. This enables you to select a different project root, so that the import wizard can mirror the parent structure when creating the new project. The example in Figure 3-12 shows the *project* folder as the parent structure to use for the new project root.

Note

If you do not use this option, resources outside the project root are linked using absolute paths.

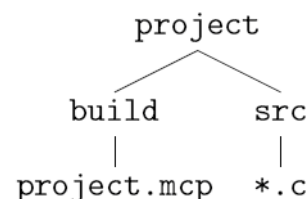


Figure 3-12 Structure with the source folder above the project file

7. In the Location field, specify the default path to create the new project in your current workspace or click **Browse...** to select an alternative location. Click **Next**.

Note

If you select **Same location as .xml file**, the new project is merged in with the existing CodeWarrior files.

8. Select the **Filter include paths** option to minimize the include paths option on the command-line, see Figure 3-13 on page 3-12. Only paths containing files ending with the specified extensions are added to the command-line options. Extensions must be given as a comma-separated list. For example: *h,inc*.

Note

If you do not use this option, all folders in the project are added as explicit includes which can be quite extensive.

9. The location of linked resources can be specified relative to a path variable, see Figure 3-13. This enables you to share projects containing linked resources without having to mirror the same file structure. If you have not specified any path variables, the **Link files relative to variable** option is disabled. Click on **Finish** to create your new project.

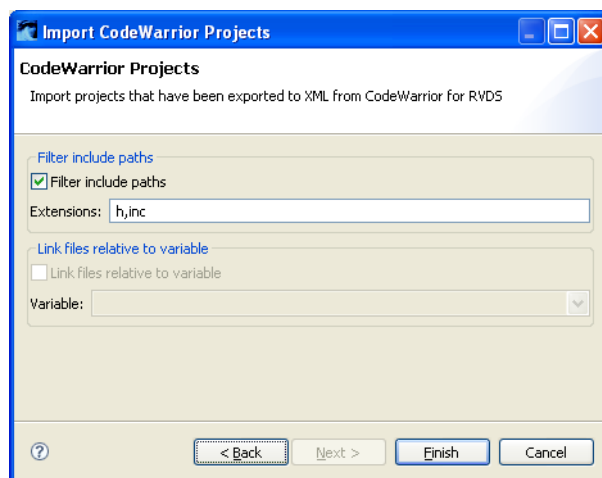


Figure 3-13 Filter include paths and path variables

Note

To add a new path variable, use the **General** → **Workspace** → **Linked Resources** option in the Preferences dialog box.

When you import a project using a path variable, the wizard automatically updates the **C/C++** → **CDT build variables** option in the Preferences dialog box to correspond with the **General** → **Workspace** → **Linked Resources** option. Both options must be synchronized with each other. If you subsequently move a resource that is linked using a path variable, you must update both options.

The imported project is visible in the Project Explorer view.

3.5 Adding a file to your project

To add a new empty source file to your project:

1. Select **File** → **New** → **File** from the main menu.
2. Select your project folder or sub-folder from the New File dialog box and enter a filename with the relevant extension in the **File name** text. See Figure 3-14.

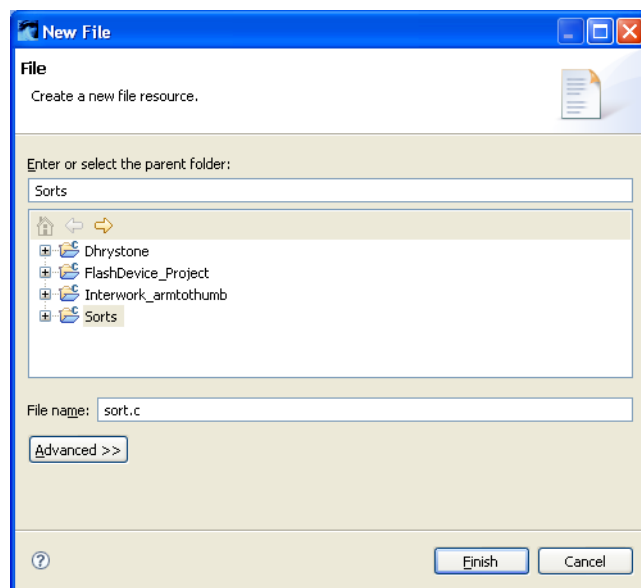


Figure 3-14 Naming your file

3. Click on **Finish**.

The new source file is visible in the Project Explorer view.

Note

- You can also create files, or drag and drop files directly into the project folder, using the file explorer. To update the views in the workbench, click on the relevant project in the Project Explorer view and select **Refresh** from the **File** menu.
 - You can also drag and drop files directly into a project folder, in the Project Explorer view in the workbench.
-

3.6 Adding a library to your project

To add a new library to your project:

1. Select your project in the Project Explorer view.
2. Select **Properties** from the Project menu.
3. Select **C/C++ Build** → **Settings** from the Properties dialog box.
4. In the Tool Settings tab, select **Directory** from the ARM Linker settings.
5. Click on the Add button in the Libraries panel to open the Add file path dialog box.
6. Click on either the **Workspace...** button or the **File system...** button to select a library file.
7. Click **OK** to close the dialog box.

The library file is now available for use by the ARM linker.

Note

The linker produces an error if the resultant file path contains spaces. This is a known issue that can be resolved by removing the quotes from the entry in the Libraries panel. Click on the **Edit** button to modify the file path.

Chapter 4

Configuring the Build and Compilation Tools

This chapter describes how to configure a build using the C/C++ Build configuration settings of the Properties dialog box. These settings determine how the ARM Compiler toolchain builds an ARM executable image or library.

Note

Build configuration settings can be applied to individual files and complete projects. To access the Properties dialog box, click on either a project folder or a file in the Project Explorer view and select **Properties** from the **File** menu.

It includes the following:

- *Accessing the build properties for an ARM project* on page 4-2
- *Accessing the build properties for a specific file* on page 4-3
- *Configuring the ARM Compiler toolchain* on page 4-4
- *Using the ARM fromelf utility* on page 4-5
- *Restoring defaults* on page 4-6.

4.1 Accessing the build properties for an ARM project

The C/C++ Build configuration panels enable you to set up the compilation tools for a specific build configuration, in your project. This section describes how to access the build configuration panels that affect all the source files in your project:

1. Select a project in the Project Explorer view. If there is no project available, add a project see Chapter 3 *Working with Projects* for more information.
2. Select **Properties** from the **Project** menu.
3. Select **C/C++ Build** → **Settings** from the Properties dialog box.
4. The Active configuration panel shows the current project type and configuration. Use the configuration drop-down menu to select either:
 - **Release**. This shows the compilation tool settings for the release build.
 - **Debug**. This shows the compilation tool settings for the debug build.

Note

New build configurations can be created by selecting **Manage** from the Active configuration panel in the Properties dialog box.

5. The compilation tools available for the project, and their respective configuration panels are displayed in the **Tool Settings** tab. See Figure 4-1.

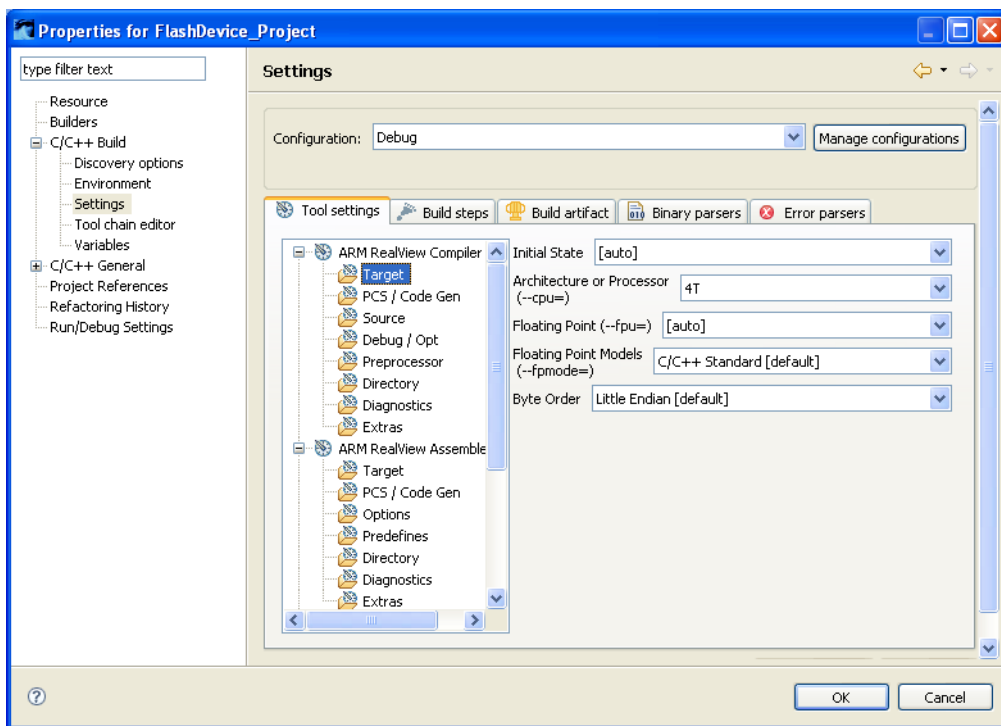


Figure 4-1 Build configuration panel for an ARM project

4.2 Accessing the build properties for a specific file

The ARM compiler and assembler can be set up differently for each source file.

Note

If you specify different options for a single file, it overrides the options specified in the project configuration panels that apply to all source files.

This section describes how to access the configuration panel for any specific source file in your project:

1. Select the source file in the Project Explorer view and select **Properties** from the File menu.
2. Select **C/C++ Build** → **Settings** from the Properties dialog box.
3. This displays the configuration settings specific to the selected file. The compilation tools are shown in the **Tool Settings** tab. If your source file has a .s extension, the **Tool Settings** tab shows the ARM Assembler settings. If your source file has a .c or .cpp extension, the **Tool Settings** tab shows the ARM Compiler toolchain settings.

4.3 Configuring the ARM Compiler toolchain

To enable you to view and set the options easily, a number of panels are provided for each of the ARM Compiler toolchain, for example, the compiler (armcc), assembler (armasm), linker (armlink), and librarian (armar). To access these panels, see *Accessing the build properties for an ARM project* on page 4-2.

Note

- The ARM Linker panels are only displayed for executable project types.
- The ARM Librarian panels are only displayed for static library project types.

See *About ARM project types* on page 3-2 for more information.

For a description of each option shown in these panels, access the dynamic help. See *Dynamic help* on page 2-22 for more information. The parent panel of each tool contains an All options field that shows all the options that you set for that tool. This list of options is passed to the compilation tool when it is invoked.

Note

- The All options field contains additional options to enable the RealView tool to work correctly in the workbench.
 - Options that are set to their compilation tool defaults, are not required by the tool when it is invoked, and so are not included in the All options field. For example, if the compiler optimization level is set to the default (level 2), the -O2 option does not appear in the All options field.
-

Each tool has an Extras panel in the **Tool Settings** tab where you can specify options that cannot be set in the other panels. The options that you set using the Extras panel override the options set in the other panels.

4.4 Using the ARM fromelf utility

The ARM fromelf utility translates *Executable and Linkable Format* (ELF) image files produced by the ARM linker into other formats suited to ROM tools and to loading directly into memory. See *ARM® Compiler toolchain Using the fromelf Image Converter* for more information on using fromelf, including information on output formats. To configure the workbench to create a plain binary file from an executable ELF image:

1. Go to the build configuration panel for your project. See *Accessing the build properties for an ARM project* on page 4-2.
2. Click on the **Build Steps** tab in the **C/C++Build** → **Settings** configuration panel.
3. In the Command field of the Post-build step, enter `fromelf --bin --output=output.bin inputfile`, where *inputfile* is the name of the ELF image. See Figure 4-2.

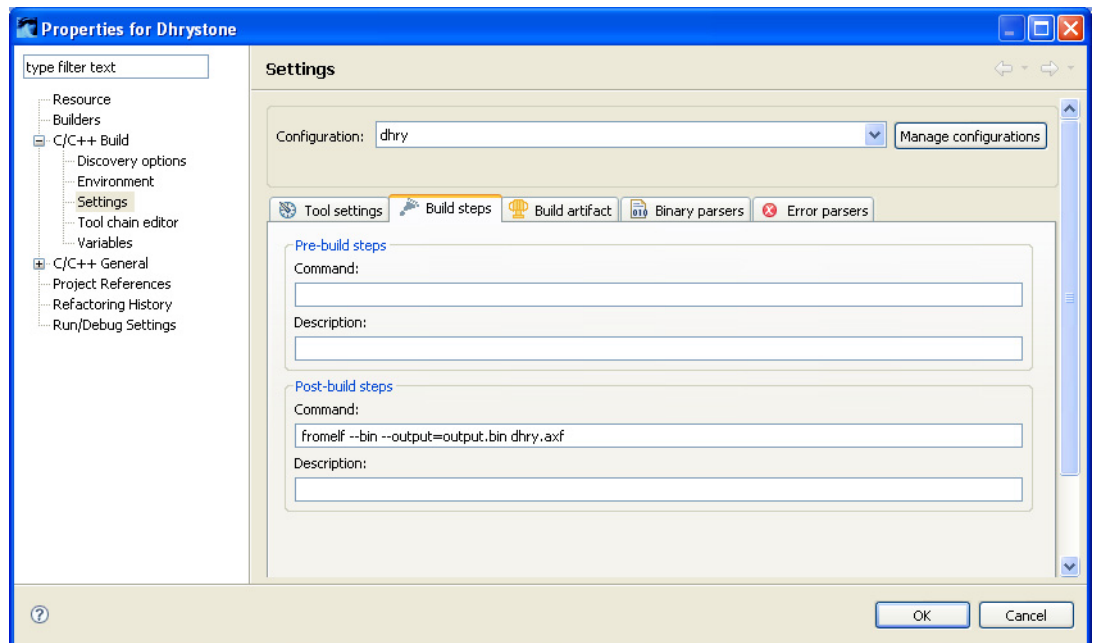


Figure 4-2 Calling fromelf from the workbench

4. The binary file gets created when you build the project and is saved as `output.bin` in the sub-folder for the active build configuration.

See also:

- *Disassembling code.*

4.4.1 Disassembling code

You can also use the ARM fromelf utility to disassemble an ELF object file and display various information. To disassemble an object file:

1. Expand your project in the Project Explorer view to show the object files.
2. Right-click on the object file in the Project Explorer view, and select **OpenWith** → **Elf Content Editor**. Alternatively, you can double-click on the object file. The disassembled object file is displayed in the default editor view. See *ELF content editor* on page 5-15 for more information.

4.5 Restoring defaults

You can use **Restore Defaults** on the configuration panel (see *Accessing the build properties for an ARM project* on page 4-2) to reset the settings of all the compilation tools displayed in the **Tool Settings** tab. The tool options revert to the defaults specific to the selected project type. It only affects the active build configuration. For example, if the active configuration is **Debug**, all the tool settings are reset only for the debug configuration. The release configuration is unaffected.

Note

- If you click on **Restore Defaults** in the project configuration panel (see *Accessing the build properties for an ARM project* on page 4-2), it does not reset any options that you set using the file specific configuration panels (see *Accessing the build properties for a specific file* on page 4-3).
 - If you click on **Restore Defaults** in a file specific configuration panel, the settings on this panel revert to the settings on the generic configuration panel. It also does not affect any options that you set using the configuration panels of other files in your project.
-

Chapter 5

Working with Editors

This chapter describes how to customize and use an editor when developing a project for an ARM target.

Editors can be associated with a specific file type by changing the default settings. You can see the default editor associations and other settings in the Preferences dialog box. See *Preferences dialog box* on page 2-14 for more information.

Double-clicking on a file in the Project Explorer view invokes the default editor.

This chapter includes the following:

- *C/C++ editor* on page 5-2
- *ARM assembler editor* on page 5-3
- *Properties editor* on page 5-4
- *Scatter file editor* on page 5-12
- *ELF content editor* on page 5-15.

5.1 C/C++ editor

The standard C/C++ editor is provided by the CDT plug-in that provides C and C++ extensions to the workbench. It provides syntax highlighting, formatting of code and content assistance when editing C/C++ code. See the *C/C++ Development User Guide* in the dynamic help for more information.

If this is not the default editor, right-click on a source file in the Project Explorer view and select **Open With → C/C++ Editor** from the context menu.

5.2 ARM assembler editor

The ARM assembler editor provides syntax highlighting, formatting of code and content assistance for labels in ARM assembly language source files. You can change the default settings in the Preferences dialog box. See *Preferences dialog box* on page 2-14 for more information.

If this is not the default editor, right-click on your source file in the Project Explorer view and select **Open With** → **ARM Assembler Editor** from the context menu.

The following shortcuts are also available for use:

Table 5-1 ARM assembler editor shortcuts

Content assist	Content assist provides auto-completion on labels existing in the active file. When entering a label for a branch instruction, Partially type the label and then use the following keyboard shortcut to display a list of valid auto-complete options: Ctrl+Space. Use the Arrow Keys to select the required label and press the Enter key to complete the term. Continue typing to ignore the auto-complete list.
Editor focus	The following options change the editor focus: <ul style="list-style-type: none"> • Outline View provides a list of all areas and labels in the active file. Click on an area or label to move the focus of the editor to the position of the selected item. • Select a label from a branch instruction and pressing F3 to move the focus of the editor to the position of the selected label.
Formatter activation	Use Ctrl+Shift+F to activate the formatter settings.
Block comments	Block comments are enabled or disabled by using Ctrl+Semicolon. Select a block of code and apply the keyboard shortcut to change the commenting state.

5.3 Properties editor

The Properties editor is provided for use with ARM assembler and C/C++ files. It enables you to configure your source code to provide GUI components for changing specific values without editing the code directly. Control tags can be embedded in commented blocks within your source code and then initialized to enable a GUI environment for setting up and modifying variables or #defines, for example, hardware initialization. Updates are synchronized so you can make changes in the source code or the GUI components.

If this is not the default editor, right-click on your source file in the Project Explorer view and select **Open With** → **Properties Editor** from the context menu.

For more information on using the properties editor, see:

- *Properties editor tabs*
- *Embedding control tags in commented blocks*
- *About control tags* on page 5-5
- *Examples of use* on page 5-7.

5.3.1 Properties editor tabs

The properties editor displays two tabs at the bottom:

Source Textual view of the source code with syntax highlighting and formatting.
You must use this tab to edit the embedded control tags that define the GUI components and associated properties.

Properties Graphical view of the embedded GUI components.

Note

You must activate the editor tabs to enable the use of GUI components. See *Embedding control tags in commented blocks* for more information.

5.3.2 Embedding control tags in commented blocks

In the **Source** tab you can embed control tags in commented blocks to produce GUI components in the **Properties** tab. Table 5-2 shows the acceptable commenting tags.

Table 5-2 Acceptable commenting tags

File Extension	Comment tag
.asm, .inc, .s	;
.c, .cpp, .h	//

Note

C style `/* */` comments are not supported.

When the control tags are correctly embedded, you can edit the source code directly using the **Source** tab or you can use the GUI components in the **Properties** tab. Both tabs are synchronized with each other when you switch between them.

To activate the tabs you must include one of the following lines of text in a commented section:

```
<<< Use Configuration Wizard in Context Menu >>>
```

or

```
<propertieseditor>
```

Marking the end of the GUI components is optional but if used, you must include one of the following lines of text in a commented section:

```
<<< end of configuration section >>>
```

or

```
<propertieseditor>
```

Coding examples are provided at the end of this section to demonstrate specific uses of the properties editor, see *Examples of use* on page 5-7.

5.3.3 About control tags

Control tags usually apply to the next item immediately following the commented section. An index can be used to modify specific items of code and ignore others that do not require modifying with GUI components.

An item can be an argument supplied to a function, a variable or `#define`. Items can be broken down into several components representing specific bit positions. Simple calculations can also be applied to component values before merging back into the relevant item by using `#` modifier tags. See Table 5-4 on page 5-6 for more information.

————— **Note** —————

The behavior is undefined if the item is a number that exceeds 2^{64} . This also applies to the arguments specified in the modifier tags.

A control tag represents a single component or a group of components. To group components together you use start and end tags. To create a GUI component you use an option tag. See Table 5-3 on page 5-6 and Table 5-4 on page 5-6 for more information.

Control tags can include extra modifier tags to define:

- range limits
- incremental step size
- size limit for a string entry
- specific bits in a number
- calculation modifiers
- group headings
- tool tips.

Control tags or modifier tags can be followed by a description, for example:

```
<h> External Bus Interface (EBI)
```

Numeric control tags that are not followed by a modifier tag, use <0x0-0xFFFFFFFF> as the default format. If a modifier tag is specified, the value of the lower limit determines the format.

Table 5-3 Control tags

Control	Description
<h>	Start tag for a group of components. Header only.
<e>	Start tag for a group of components. Header with check box option to enable or disable the following group. For example: <e> the next item of code contains the enable/disable value. Index 0 being the default. <e.4> bit 4 in the next item of code contains the enable/disable value. <e1.4> index 1 specifies the second item of code with bit 4 contains the enable/disable value.
</h> or </e>	End tag for a group of components.
<q>	Check box option. For example: <q> the next item of code contains the enable/disable value. Index 0 being the default. <q.4> bit 4 in the next item of code contains the enable/disable value. <q1.4> index 1 specifies the second item of code with bit 4 contains the enable/disable value.
<o>	Combo box or spin box option. For example: <o> the next item of code contains the value. Index 0 being the default. <o.4> bit 4 in the next item of code contains the value. <o1.4..6> index 1 specifies the second item of code with bits 4 to 6 containing the values.
<s>	Text box option. Size limit is optional. For example: <s> the next item of code contains the string. <s.10> the next item of code contains the string with a size limit of 10 characters.

Table 5-4 Modifier tags

Modifier	Description
<i>	Tool tip tag to provide help on the previous component. Use on start tags or option tags.
<0-31>	Produces a spin box containing a range of decimal values. For example, a range of 0-31.
<0-100:10>	Produces a spin box containing a range of decimal values with increments defined by the step value. For example, a range of 0-100 with a step of 10.

Table 5-4 Modifier tags (continued)

Modifier	Description
<0x40-0x1000:0x10>	Produces a spin box containing a range of hex values with increments defined by the step value. For example, a range of 0x40-0x1000 with a step of 0x10. ———— Note ———— <0x0-0xFFFFFFFF> is the default if no modifier tag is specified.
<0=> <i>First_Description</i>	Produces a combo box containing the textual description. This example produces <i>First_Description</i> as the first entry in the combo box. The value 0 is merged back into the code.
<#+1> <#-1> <#*8> <#/3>	Applies a calculation before merging back into code. The following calculations are available: add, subtract, multiply, and divide.

5.3.4 Restrictions of use

The following restrictions apply:

- If two options control the same value, the last change is applied
- Whitespace characters are not permitted inside a tag
- Strings range from 0 to a maximum of 255 characters
- Bit values range from 0 to a maximum of 31
- Numbers range from 0 to a maximum of 2^{64}
- An error occurs if the spin box range exceeds:
 - 2^{32} for hex
 - 2^{31} for decimal
 - The behavior is undefined if the item is a number that exceeds 2^{64} . This also applies to the arguments specified in the modifier tags.

5.3.5 Examples of use

The following illustrated examples show incomplete extracts of source code with various embedded control tags and the resultant GUI components:

Bus configuration example

Example 5-1 on page 5-8 shows control tags for modifying the second argument of the following function:

```
WRITE_MEM(0xFFE00000, 0x2034A9)
```

This is defined by index 1 in the <e1.x> and <o1.x> control tags.

———— **Note** ————

Use index 0, for example, <e0.x> if you want to embed control tags for the first argument of the following function:

```
WRITE_MEM(0xFFE00000, 0x2034A9)
```


Example 5-1 Bus configuration control tags

```
//C example
//*****
// <propertieseditor>
//*****
// <h> External Bus Interface (EBI)
//   <e1.13> CS0: Enable Chip Select 0
//     <o1.20..31> BA: Base Address <0x0-0xFFFF0000:0x100000> <#/0x100000>
//       <i> Start Address for Chip Select Signal
//
//   NOTE: Base Address works with bits 20 to 31 of second argument producing
//   spin box with range and step values. Applies calculation before parsing.
//
//   <o1.7..8>   PAGES: Page Size      <0=> 1M Byte   <1=> 4M Bytes
//                                     <2=> 16M Bytes  <3=> 64M Bytes
//
//               <i> Selects Active Bits in Base Address
//   <o1.0..1>   DBW: Data Bus Width   <1=> 16-bit   <2=> 8-bit
//   <o1.12>     BAT: Byte Access Type <0=> Byte-write
//                                     <1=> Byte-select
//
//   <e1.5>     WSE: Enable Wait State Generation
//     <o1.2..4> NWS: Number of Standard Wait States <1-8> <#-1>
//
//   </e>
//   <o1.9..11> TDF: Data Float Output Time <0-7>
//               <i> Number of Cycles Added after the Transfer
//
//   </e>
// </h>
WRITE_MEM(0xFFE00000, 0x2034A9); // EBI_CS0: Flash
//*****
// </propertieseditor>
//*****
```

Figure 5-1 shows the GUI components produced by this source code.

Option	Value
External Bus Interface (EBI)	
CS0: Enable Chip Select 0	<input checked="" type="checkbox"/>
BA: Base Address	0x200000
PAGES: Page Size	4M Bytes
DBW: Data Bus Width	16-bit
BAT: Byte Access Type	Byte-select
WSE: Enable Wait State Generation	<input checked="" type="checkbox"/>
TDF: Data Float Output Time	2

Figure 5-1 Bus configuration GUI components

As a guideline, the control tags embedded in this example include symbol names in front of each description. Figure 5-2 shows these symbol names and associates them with their bit positions.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
...binary								0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	0	1											
...hex								2				0				3				4				A				9														
BA																			C	S	R	O	B	A	T	T	D	F	P	A	G	E	S	W	S	E	N	W	S	D	B	W

Figure 5-2 Bus configuration bit positions

Password string example

The sample code in Example 5-2 results in the GUI components shown in Figure 5-3.

Example 5-2 Password string control tags

```
//C example
//*****
// <propertieseditor>
//*****
// <o> Program Entry Point
PC = 0x4000000;

// <s> Change ID
// <s1.30> Change Password String
#define ID "My User ID"
char pw[] = "My Password";
//*****
// </propertieseditor>
//*****
```

Option	Value
Program Entry Point	0x4000000
Change ID	My User ID
Change Password String	My Password

Figure 5-3 Password string GUI components

Data read protocol example

The sample code in Example 5-3 results in the GUI component shown in Figure 5-4.

Example 5-3 Data read protocol control tags

```
//C example
//*****
// <propertieseditor>
//*****
// <q1.4> DRP: Data Read Protocol
// <0=> Standard Read
// <1=> Early Read
DATAREAD(0xFFE00024, 0x10); // EBI_MCR: Data Read Protocol
//*****
// </propertieseditor>
//*****
```

Option	Value
DRP: Data Read Protocol	<input checked="" type="checkbox"/>

Figure 5-4 Data read protocol GUI component

Stack configuration example

The sample code in Example 5-4 results in the GUI components shown in Figure 5-5.

Example 5-4 Stack configuration control tags

```
; Assembler example
;*****
; <propertieseditor>
;*****
; <h> Stack Configuration (Stack Sizes in Bytes)
;   <o0> Undefined Mode      <0x0-0xFFFFFFFF0:8>
;   <o1> Supervisor Mode    <0x0-0xFFFFFFFF0:8>
;   <o2> Abort Mode         <0x0-0xFFFFFFFF0:8>
;   <o3> Fast Interrupt Mode <0x0-0xFFFFFFFF0:8>
;   <o4> Interrupt Mode     <0x0-0xFFFFFFFF0:8>
;   <o5> User/System Mode   <0x0-0xFFFFFFFF0:8>
; </h>

UND_Stack_Size EQU    0xe0
SVC_Stack_Size EQU    0x1000
ABT_Stack_Size EQU    0xa8
FIQ_Stack_Size EQU    0x40
IRQ_Stack_Size EQU    0x68
USR_Stack_Size EQU    0x3f8

Stack_Size      EQU    (UND_Stack_Size + SVC_Stack_Size + ABT_Stack_Size + \
                        IRQ_Stack_Size + IRQ_Stack_Size + USR_Stack_Size)

Stack_Mem       AREA   STACK, NOINIT, READWRITE, ALIGN=4
                SPACE  Stack_Size

Stack_Top       EQU    Stack_Mem + Stack_Size
;*****
; </propertieseditor>
;*****
```

Option	Value
<input checked="" type="checkbox"/> Stack Configuration (Stack Sizes in Bytes)	
Undefined Mode	0xE0
Supervisor Mode	0x1000
Abort Mode	0xA8
Fast Interrupt Mode	0x40
Interrupt Mode	0x68
User/System Mode	0x3F8

Figure 5-5 Stack configuration GUI components

VPBDIV definitions example

The sample code in Example 5-5 results in the GUI components shown in Figure 5-6.

Example 5-5 VPBDIV definitions control tags

```
; Assembler example
*****
; <propertieseditor>
*****
; VPBDIV definitions
VPBDIV      EQU      0xE01FC100      ; VPBDIV Address

; <e> VPBDIV Setup
; <i> Peripheral Bus Clock Rate
;
;   NOTE: Option tag <e> uses index 0 as default if not specified
;         Enables or disables group of options using setup variable
;
;   <o1.0..1> VPBDIV: VPB Clock
;             <0=> VPB Clock = CPU Clock / 4
;             <1=> VPB Clock = CPU Clock
;             <2=> VPB Clock = CPU Clock / 2
;   <o1.4..5> XCLKDIV: XCLK Pin
;             <0=> XCLK Pin = CPU Clock / 4
;             <1=> XCLK Pin = CPU Clock
;             <2=> XCLK Pin = CPU Clock / 2
; </e>
VPBDIV_SETUP EQU      0
VPBDIV_Val   EQU      0x20
*****
; </propertieseditor>
*****
```

Figure 5-6 shows some of the GUI components greyed out because the main group header is disabled. To enable the group you can either click on the GUI box for VPBDIV Setup or edit the relevant code as follows:

```
VPBDIV_SETUP EQU      1
```

Option	Value
<input checked="" type="checkbox"/> VPBDIV Setup	<input type="checkbox"/>
VPBDIV: VPB Clock	VPB Clock = CPU Clock / 4
XCLKDIV: XCLK Pin	XCLK Pin = CPU Clock / 2

Figure 5-6 VPBDIV definitions GUI components

5.4 Scatter file editor

The scatter file editor enables you to easily create and edit scatter-loading description files for use with the ARM linker to construct the memory map of an image. It provides a text editor, a hierarchical tree and a graphical view of the regions and output sections of an image. You can change the default syntax formatting and color schemes in the Preferences dialog box. See *Preferences dialog box* on page 2-14 for more information.

If this is not the default editor, right-click on your source file in the Project Explorer view and select **Open With** → **Scatter File Editor** from the context menu.

The scatter file editor displays the following tabs:

Source Textual view of the source code with syntax highlighting and formatting.

Regions/Sections

A graphical view showing load and execute memory maps. These are not editable, however, you can select a load region to show the related memory blocks in the execution regions.

The scatter file editor also provides a hierarchical tree with associated toolbar and context menus using the Outline view. Clicking on a region or section in the Outline view moves the focus of the editor to the relevant position in your code. If this view is not visible, select **Show View** → **Outline** from the Window menu.

Note

See *ARM® Compiler toolchain Using the Linker* and *ARM® Compiler toolchain Linker Reference* for more information on how to use scatter-loading description files.

The scatter file editor does not support preprocessing directives, for example:

```
#! armcc -E.
```

Before you can use a scatter-loading description file you must add the `--scatter=file` option to the project within the **C/C++ Build** → **Settings** → **Tool settings** → **ARM RealView Linker** → **Output** panel of the Properties dialog box. See *Properties dialog box* on page 2-15 for more information.

See also:

- *Example of a scatter-loading description file.*

5.4.1 Example of a scatter-loading description file

To create a scatter-loading description file:

1. Use an existing project or create a new project. See Chapter 3 *Working with Projects* for more information.
2. Create a new project and add a new empty text file with the extension `.scat`, for example `scatter.scat`. See *Creating a new RealView project* on page 3-5 and *Adding a file to your project* on page 3-13 for more information.
3. Using the Outline view, click on the toolbar icon or right-click and select **Add load region** from the context menu.

The **Add load region** dialog box is displayed. Figure 5-7 on page 5-13 shows an example:

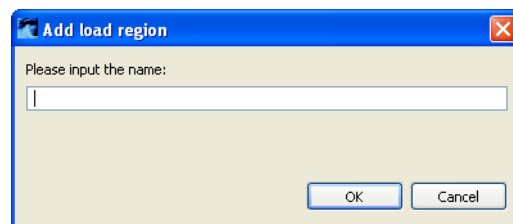


Figure 5-7 Add load region name

4. Enter a load region name, for example, LR1.
5. Click **OK**.
6. Modify the load region as shown in Example 5-6:

Example 5-6 Simple scatter-loading description file

```
LR1 0x0000 0x8000
{
    LR1_er1 0x0000 0x8000
    {
        * (+R0)
    }

    LR1_er2 0x10000 0x6000
    {
        * (+RW,+ZI)
    }
}
```

This example shows a simple scatter-loading description file

7. Save your changes.
8. To set up a more complex scatter-loading file:
 - a. Repeat steps 3 to 5 to add a second load region LR2.
 - b. Modify the LR1 and LR2 load regions as shown in Example 5-7.

Example 5-7 Complex scatter-loading description file

```
LR1 0x0
{
    LR1_er1 0x0
    {
        program.o (+R0)
    }

    LR1_er2 0x18000 0x8000
    {
        program1.o (+RW,+ZI)
    }
}

LR2 0x4000
{
    LR2_er1 0x4000
    {
        program2.o (+R0)
    }
}
```

```
}  
  
LR2_er2 0x8000 0x8000  
{  
    program2.0 (+RW,+ZI)  
}  
}
```

- c. Save your changes.

5.5 ELF content editor

The ELF content editor creates forms and graphical views for the selected ELF file using specific `fromElf` command-line options to generate the relevant data. You can use this editor to view the contents of image files, object files and library files. The editor is read-only and cannot be used to modify the contents of any files.

If this is not the default editor, right-click on your source file in the Project Explorer view and select **Open With** → **ELF Content Editor** from the context menu.

The ELF content editor displays one or more of the following tabs depending on the selected file type:

Overview An image or object file shows header information and section details.
 A library file shows a breakdown of the data types for each object within the selected library file.

Symbol Table

Tabular view showing the breakdown of all symbols. The data can be saved to a *Comma Separated Value* (CSV) file.

This tab is only available for an image or object file.

Instruction Sizes

Graphical view of instruction usage and functional types. The data can be saved to a CSV file.

This tab is only available for an image or object file.

Disassembly

Textual view with syntax highlighting.

This tab is only available for an image or object file.

See also:

- *Overview*
- *Symbol table* on page 5-18
- *Instruction sizes* on page 5-19
- *Disassembly* on page 5-20.

5.5.1 Overview

The **Overview** tab displays different information depending on the selected file type. An image or object file shows the header information and section details for each section in the input file. A library file shows a breakdown of the data types for each object within the selected library file.

Image or object file

For an image or object file, the **Overview** tab uses `fromelf --text -v` command-line options to provide a form view of the ELF header and section information.

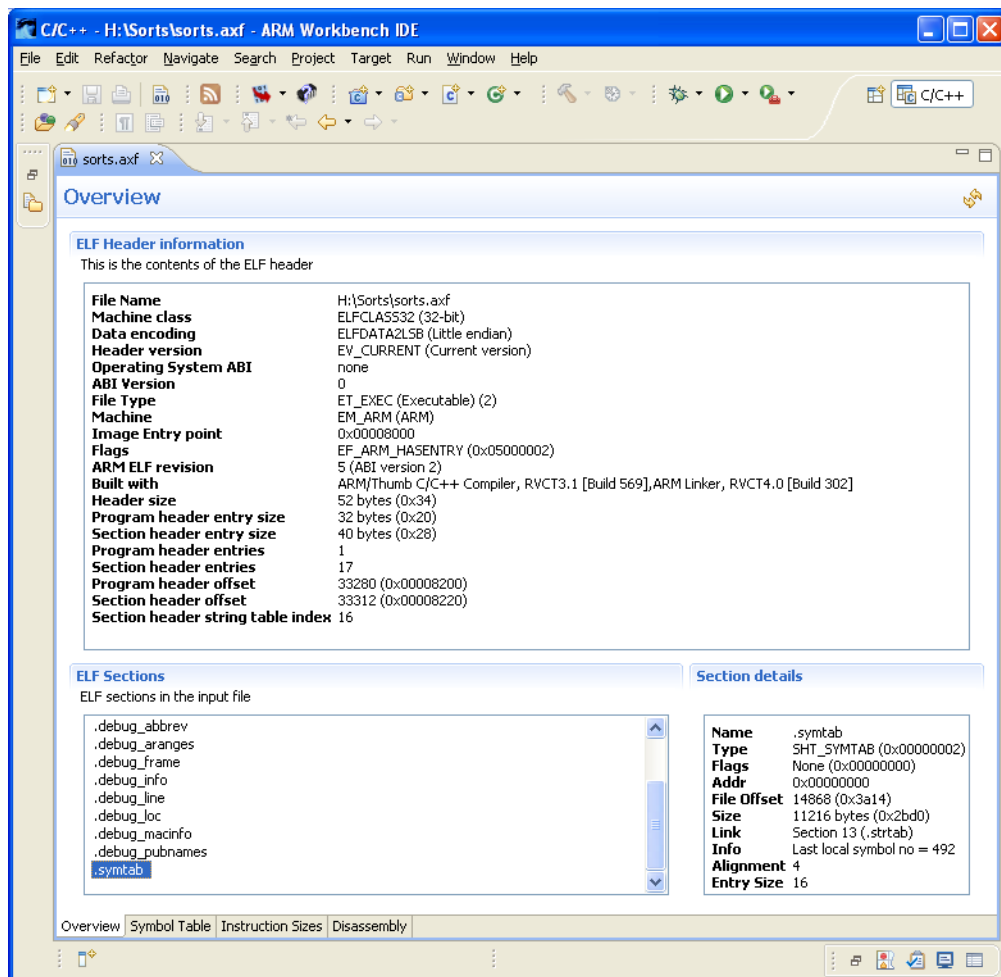


Figure 5-8 Overview tab

Library file

For a library file, the **Overview** tab uses `armar --sizes` command-line option to provide a breakdown of each object within the selected library file. The data can be displayed as a bar chart or a pie chart and it can also be saved to a CSV file.

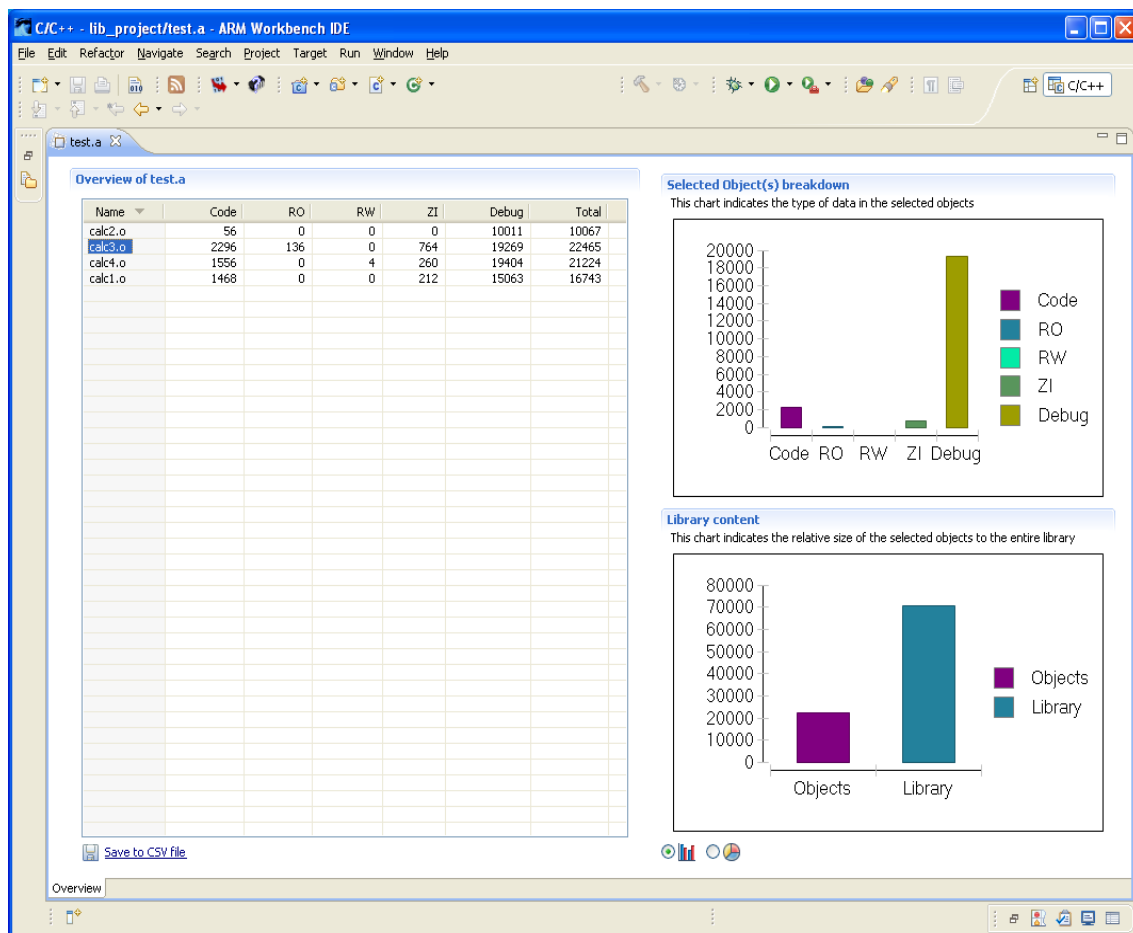


Figure 5-9 Overview tab

5.5.2 Symbol table

For an image or object file, the **Symbol Table** tab uses `fromelf --text -s` command-line options to provide a tabular view of the symbols. You can sort the data by columns and also use the filter option to hide the mapping symbols. The data can be saved to a CSV file.

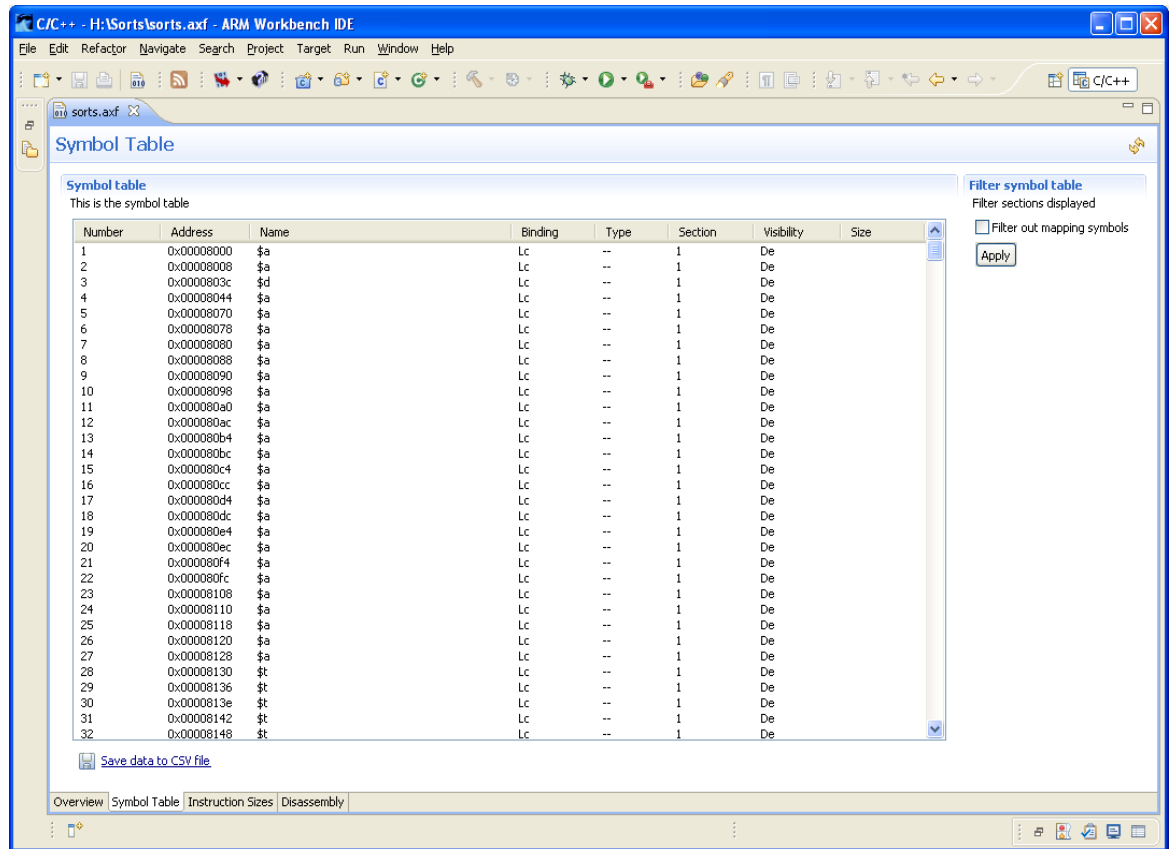


Figure 5-10 Symbol Table tab

5.5.3 Instruction sizes

For an image or object file, the **Instruction Sizes** tab uses `fromelf --text --info=instruction_usage` command-line options to provide a categorized view of all instructions.

The data can be displayed as a bar chart, pie chart or tabular format. It can also be saved to a CSV file.

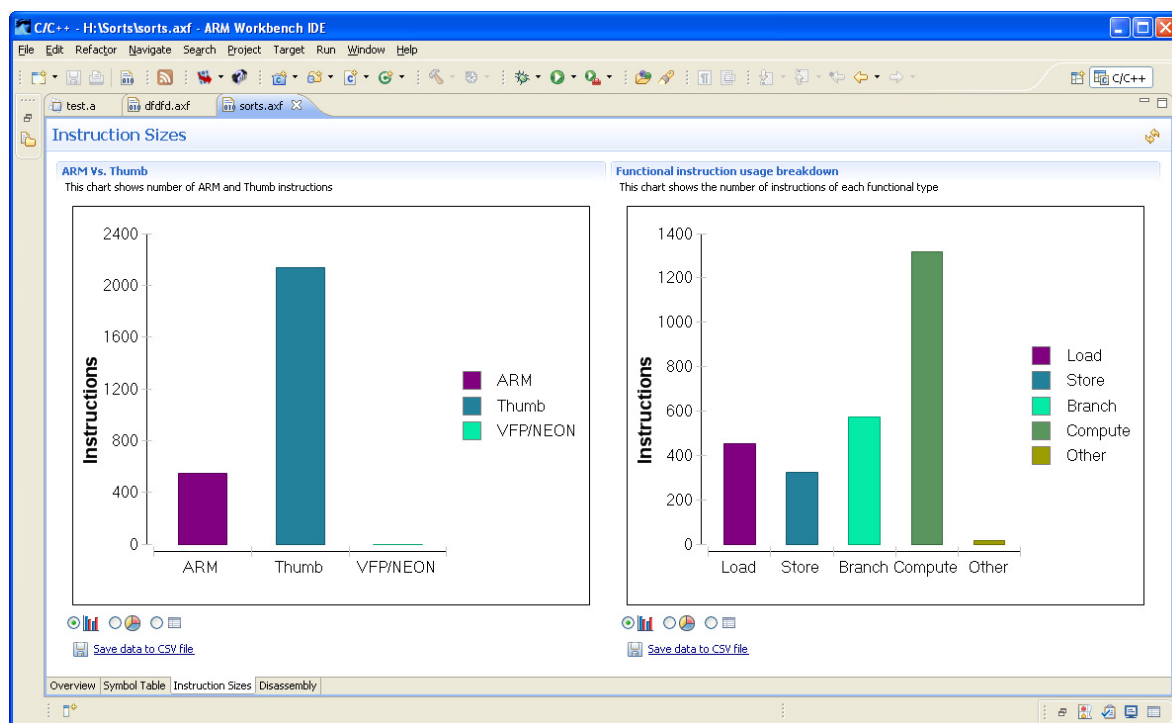


Figure 5-11 Instruction Sizes tab

5.5.4 Disassembly

For an image or object file, the **Disassembly** tab uses `fromelf --text -c` command-line options to display the output with syntax highlighting. The color schemes and syntax preferences use the same settings as the ARM assembler editor. There are several keyboard combinations that can be used to navigate around the output:

- Use **Ctrl+F** to open the Find dialog box to search the output
- Use **Ctrl+Home** to move the focus to the beginning of the output
- Use **Ctrl+End** to move the focus to the end of the output.

You can also use the **Copy** and **Find** options in the context menu by right-clicking in the Disassembly view.

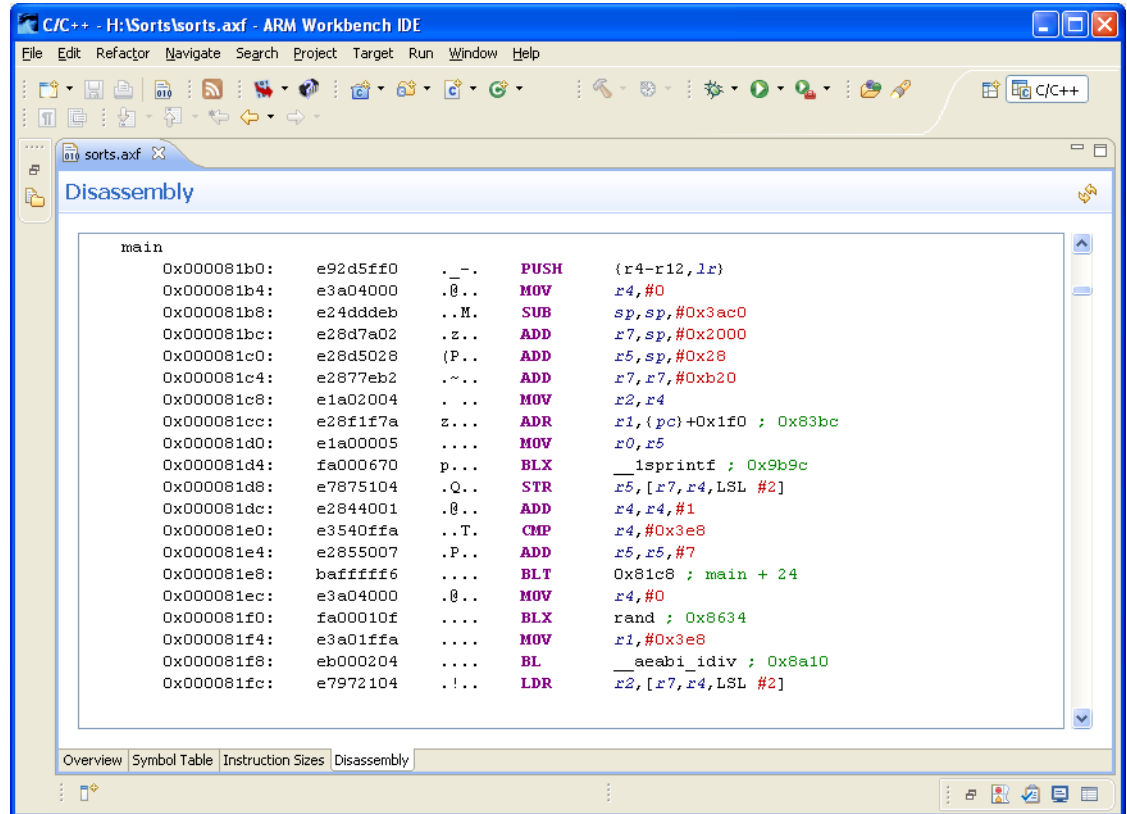


Figure 5-12 Disassembly tab

Chapter 6

Working with the ARM Flash Programmer

This chapter describes how to use the ARM® flash programmer to control and update flash memory on your chosen target.

It includes the following:

- *About the ARM flash programmer* on page 6-2
- *Programming a flash device* on page 6-4
- *Importing a flash image* on page 6-6
- *Managing flash targets* on page 6-7
- *Using the flash device manager* on page 6-9
- *Creating a new flash algorithm* on page 6-11
- *Exporting a board for use with RealView Debugger* on page 6-14
- *Exporting a flash device for use with RealView Debugger* on page 6-15.

6.1 About the ARM flash programmer

The ARM flash programmer connects to your target using DSTREAM or RealView ICE to download a flash algorithm and your flash image into RAM using as much memory as required. You can specify the base address and size of RAM using target configurations. Flash algorithms can be modified to suit specific flash devices or you can create your own. The image is copied from RAM to your flash device(s) when the algorithm executes.

To send an image to your flash device, you must set up the following:

- flash image, Table 6-1 lists the supported image types
- flash programmer target configuration
- DSTREAM or RealView ICE.

Table 6-1 Mapping extensions to image types

Extension	Image Type
.bin	Binary image
.axf, .elf	ELF image
.i32	Intel Hex-32
.m32	Motorola 32-bit S-record
.vhx	Byte Oriented (Verilog Memory Model) Hex

The main ARM flash programmer features are:

- The Send To dialog box enables you to send an image to a specific memory location on your flash device. See *Programming a flash device* on page 6-4 for more information.
- The Manage targets... dialog box enables you to set up a flash target configuration that connects with your device. See *Managing flash targets* on page 6-7 for more information.
- The Flash Device Manager dialog box enables you to add new flash algorithms to your workspace. See *Using the flash device manager* on page 6-9 for more information.
- The New Flash Device Project wizard enables you to create a new flash algorithm for use in an ARM project or for export and distribution to a 3rd party. See *Creating a new flash algorithm* on page 6-11 for more information.

The flowchart in Figure 6-1 on page 6-3 illustrates these features and shows how they interconnect.

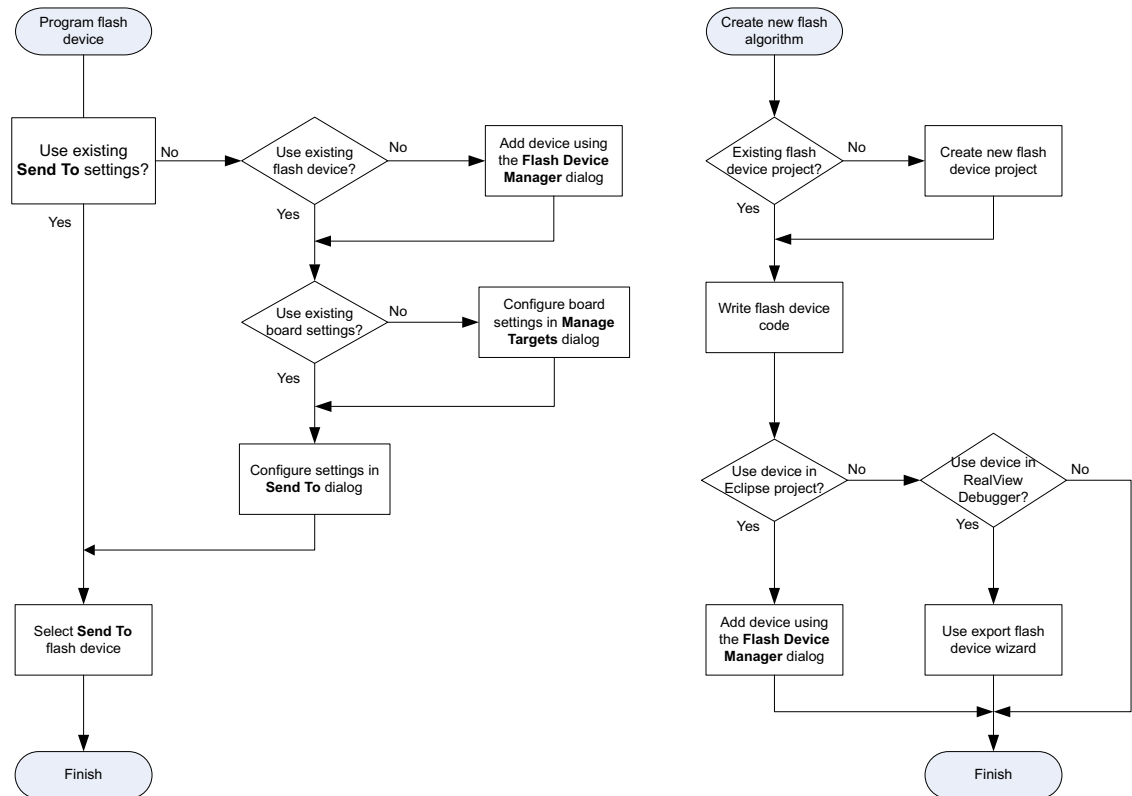


Figure 6-1 The ARM flash programmer

6.2 Programming a flash device

To program flash memory on your device, choose one of the following options:

- If you have previously set up a flash programmer configuration for your device, select **Run** → **Send To** → **Send To** → **Flash Image** from the main menu.
- If your flash programmer configuration is not set up:
 1. Click on **Run** → **Send To** → **Send To...** from the main menu.
 2. Use the tabs in the Send To dialog box to complete this process. An example is shown in Figure 6-2.

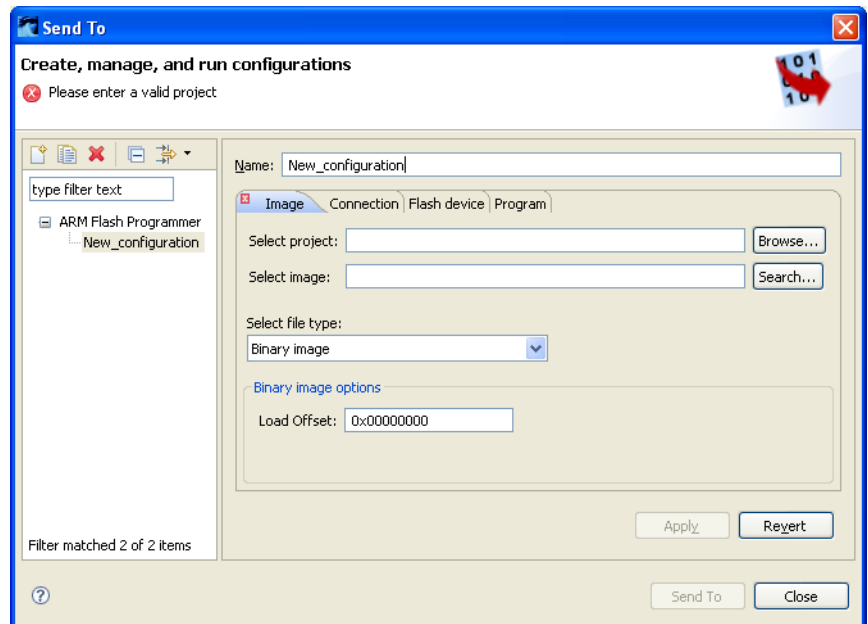


Figure 6-2 Create, manage, and run configurations

The tabs are:

Image Use to select the image to send to your flash device. The flash image must exist in an ARM project or project sub-folder before you can select it in this tab.

Connection

Use to set up the connection method for your flash device. Select your target if you have already configured one or select **Add new RealView ICE target** from the toolbar and click **Configure...** to set up the connection properties.

Flash device

Use to set up the target details for your flash device. Select your target configuration and then select the relevant flash devices, either one NAND or all the NOR devices.

Program

Use to set up the erase method and verification for your flash device.

Click on **Apply** to save the current configuration but not connect to your flash device.

Click on **Revert** to undo any changes and revert to the last saved configuration.

Click on **Send To** to connect to your flash device and send the selected flash image.

For more information on the options available in each of the tabs, use the dynamic help. See *Dynamic help* on page 2-22 for more information.

3. Click on **Send To** to send your flash image to your flash device.

———— **Note** ————

Programming your flash device can take a long time!

The Program Devices dialog box and the main Console view, update as each programming operation completes. On completion, the results are displayed in the Flash Programmer Results dialog box as shown in Figure 6-3.

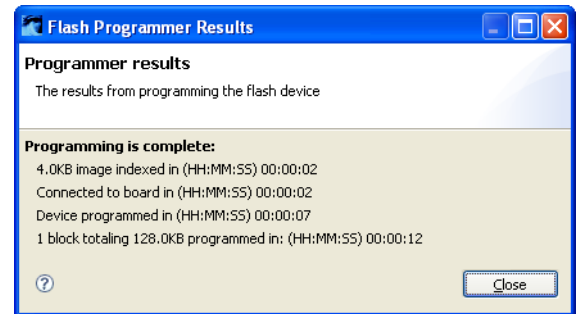


Figure 6-3 Results from programming your device

6.3 Importing a flash image

To use the ARM flash programmer, a flash image must exist in an ARM project or project sub-folder.

This section describes how to import a flash image from an external folder into an existing project within the current workspace.

1. Select **Import...** from the **File** menu and then select **Flash Programmer** → **Flash Image**, see Figure 6-4. Click on **Next**.

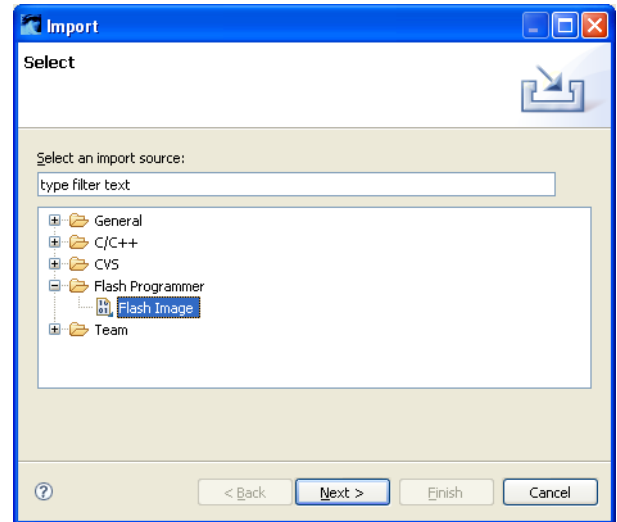


Figure 6-4 Select flash image to import

2. Select the image source, the destination project and also the relevant import options. Click on **Finish**.

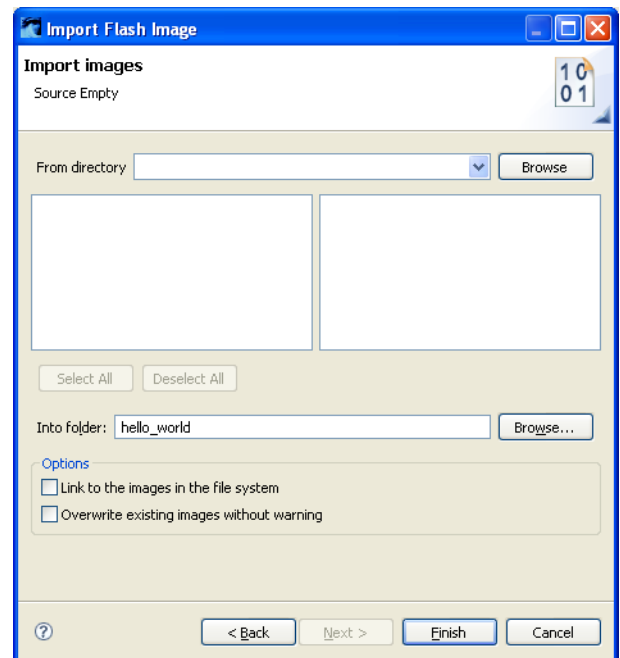


Figure 6-5 Flash image import settings

6.4 Managing flash targets

To access a flash device from the workbench you must set up a flash target configuration. A flash target configuration can have multiple flash devices associated with it. The ARM flash programmer plug-in provides several built-in configurations and more can be created.

To edit or view a flash target configuration, select **Manage Targets...** from the **Target** menu. Figure 6-6 shows the Manage Targets dialog box.

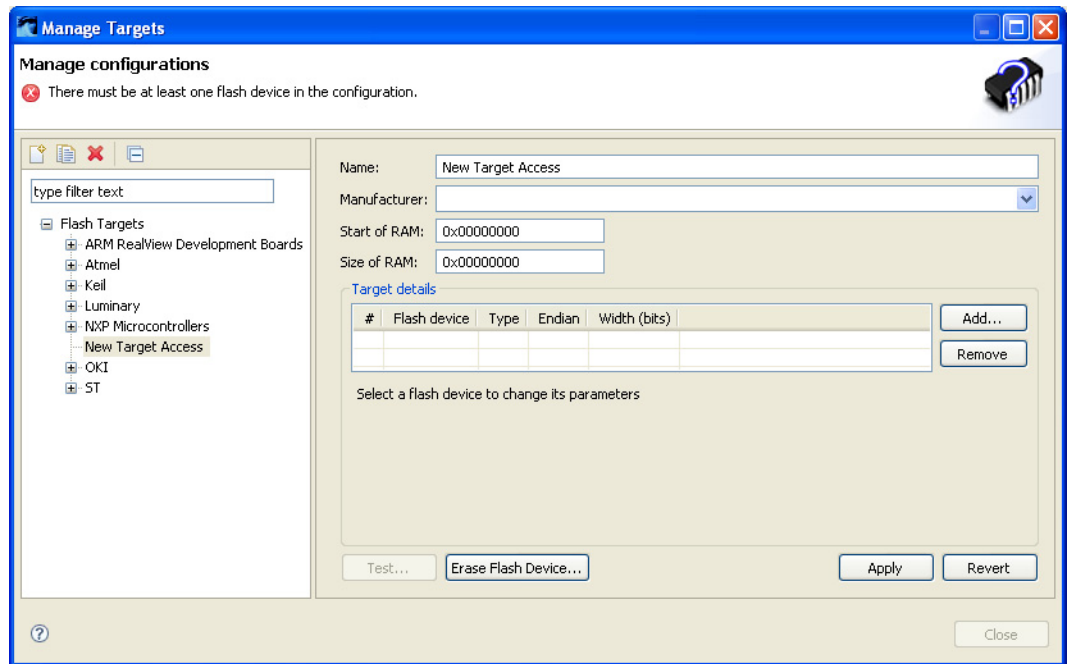


Figure 6-6 Manage configurations

1. Select your flash target configuration from the tree. To create a new configuration, click on the toolbar or right-click on an existing configuration, and select **New** or **Duplicate**.
2. Set up the target details table and flash device parameters as required to complete this process. For information on each configuration option, access the dynamic help. See *Dynamic help* on page 2-22 for more information.
Click on **Apply** to save the current configuration. This does not connect to your flash device.
Click on **Revert** to undo any changes and revert to the last saved configuration.
3. To test your device, Click on **Test...** to access the Test Target dialog box as shown in Figure 6-7 on page 6-8.
 - a. Click on **Configure...** to set up the **Connection method** and **CPU** settings if you have not already set them up.
 - b. Click on **Test device...** to erase, program, and validate the required blocks.

————— Note —————

Testing your flash device can take a long time.

The Progress Information dialog box and the main Console view update as each programming operation completes. On completion, the results are displayed in the Flash Programmer Results dialog box as shown in Figure 6-8 on page 6-8.

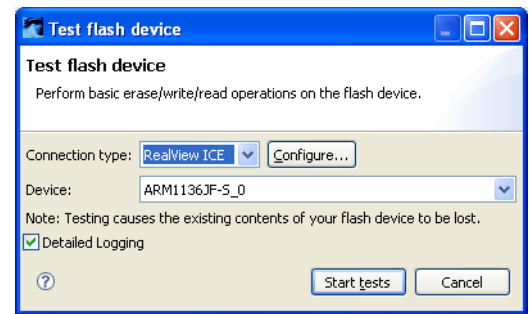


Figure 6-7 Test target connection method

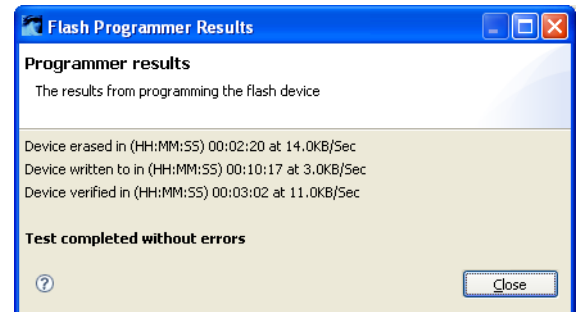


Figure 6-8 Results from testing your device

6.5 Using the flash device manager

A new flash algorithm must exist in the current workspace before you can access it from the Manage Targets dialog box or export it for use with RealView Debugger.

This section describes how to add a new flash algorithm to your workspace.

1. Select **Flash Device Manager** from the **Target** menu, see Figure 6-9.

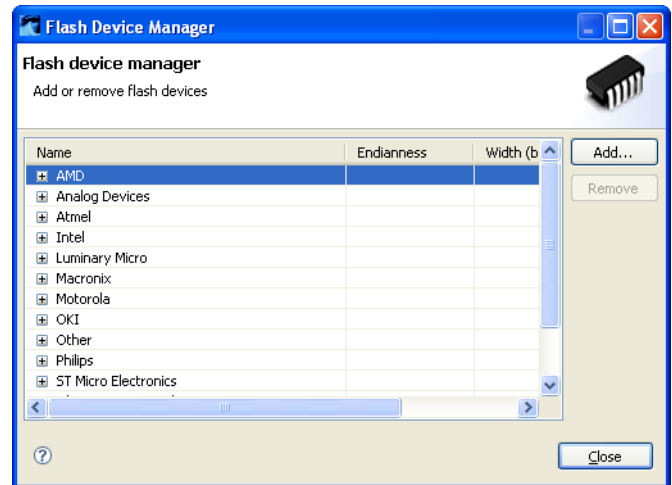


Figure 6-9 Add or remove flash devices

2. Click on **Add...** to open the Import Flash Device dialog box and add your new flash algorithm to your workspace.

Note

To remove a flash algorithm, select the name of your algorithm and click on **Remove**. You cannot remove a built-in flash algorithm.

3. Select the project containing your flash algorithm, see Figure 6-10.
Click on **Next**.

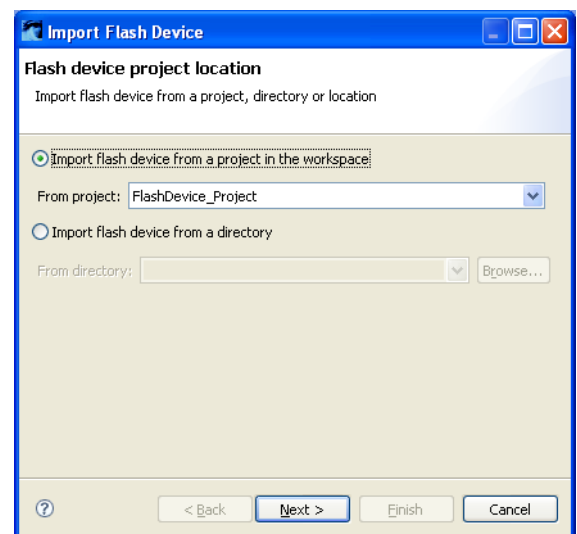


Figure 6-10 Import a flash device

4. Select a build configuration and the relevant object file associated with your flash device. It is recommended that you select a fully optimized build, for example, the Release configuration, resulting in faster flash operations. Click on **Finish**.

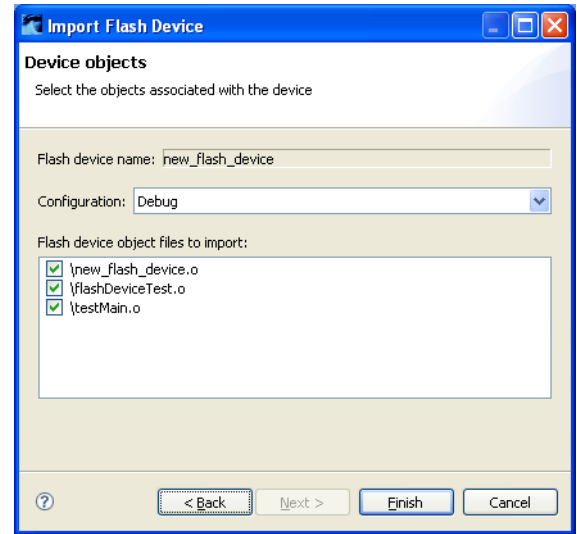


Figure 6-11 Import flash device object files

5. Your flash algorithm now appears in the Flash Device Manager dialog box and is available for use by the ARM flash programmer. Click on **Close**.

6.6 Creating a new flash algorithm

Flash device projects are used to create new flash algorithms for distribution to a 3rd party or for importing into another ARM project. The flash device project wizard creates a new project folder containing several .h and .c files.

This section describes how to create a new flash algorithm:

1. Select **File** → **New** → **Project...** from the main menu.
2. Select **Flash Programmer** → **New RealView Flash Device Project**, see Figure 6-12. Click on **Next**.

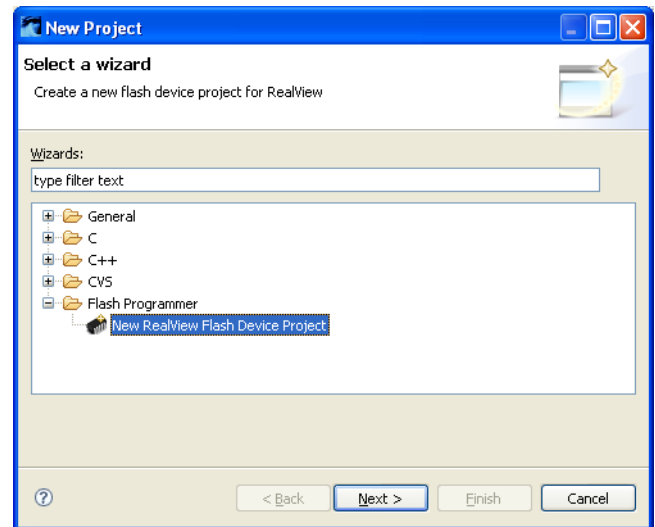


Figure 6-12 New flash device project

3. Enter a name for your flash device project, see Figure 6-13.

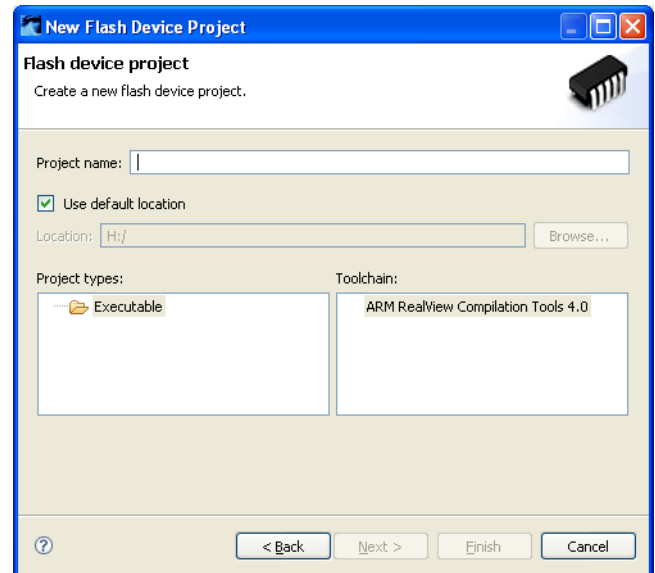


Figure 6-13 Naming your flash project

4. Leave the **Use default location** option selected so that the project is created in the default folder shown. Alternatively, deselect this option and browse to your preferred project folder. Click on **Next**.

5. Enter the flash device details for your project as appropriate, see Figure 6-14. Ensure you enter a meaningful name in the Flash Device Name field. Click on **Next**.

Figure 6-14 Flash device details

Note

These flash device details can be modified later using the Flash Device selection in the Properties dialog box for your project.

6. In the Configurations panel, leave the **Debug** and **Release** options selected to enable you to save different project settings for both debug and release configurations, see Figure 6-15 on page 6-13. Click on **Finish**.

Note

If **Build Automatically** is selected, the project builds automatically when you click on **Finish**. If not, you must select **Build Project** from the **Project** menu to complete the build step.

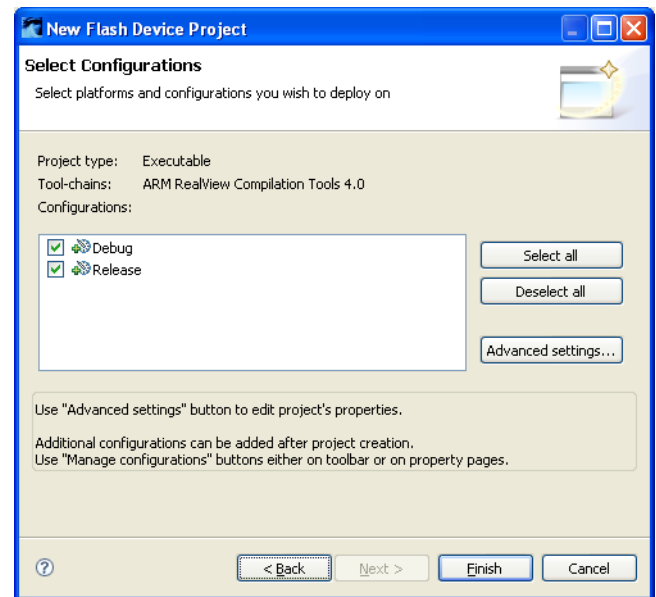


Figure 6-15 Flash device configuration settings

The new project is visible in the Project Explorer view. The flash device project wizard creates the following files and places them in your project folder:

XXXXX.c

Template .c file created with the same name as you entered for the flash device. You must edit this file to implement your flash algorithm.

See *Application Note 190 Creating Flash Algorithms with Eclipse* for more information.

flashDeviceTest.h

Header file declaring the test harness. Do not modify this file.

flashDeviceTest.c

Implementation of the test harness. You can step through this code in a debugger to debug your flash algorithm. You can edit some options within this file to change the behavior of the tests.

testMain.c

Defines a main() function used when linking the Debug configuration of your flash algorithm. The main() function calls the test harness, passing in the address of your flash device. Do not modify this file.

dummyMain.c

Defines a main() function used when linking the Release configuration of your flash algorithm. Do not modify this file.

6.7 Exporting a board for use with RealView Debugger

New flash device boards created with the ARM Flash Programmer can be exported for use with RealView Debugger. This section describes how to create the required *Board Chip Definition* (BCD) file.

1. Select **Export...** from the **File** menu and then select **Flash Programmer** → **RealView Debugger Board**, see Figure 6-16. Click on **Next**.

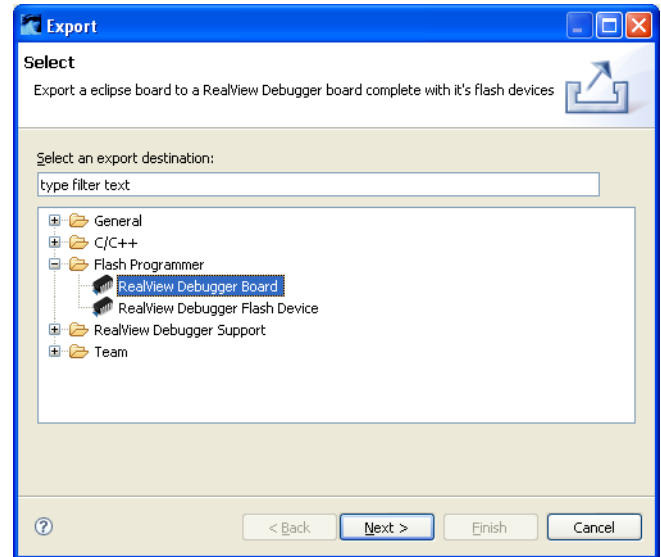


Figure 6-16 Export RealView Debugger Board

2. Select the required board and device, see Figure 6-17. Click on **Finish** to export the selected board and device to RealView Debugger.

The resultant BCD file is exported to the RealView Debugger default folder:

`install_directory\RVD\Core\...\etc`

On launching RealView Debugger, the new board can be selected in the Connection Properties window. See the *RealView® Debugger Target Configuration Guide* for more information.

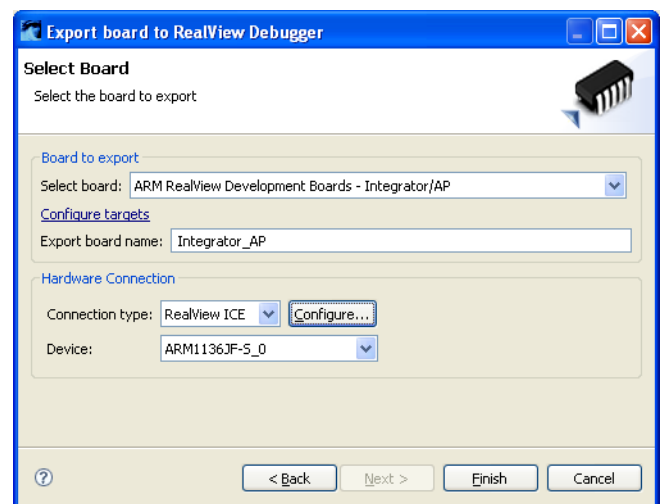


Figure 6-17 Select board configuration

6.8 Exporting a flash device for use with RealView Debugger

New flash algorithms created with the ARM Flash Programmer can be exported for use with RealView Debugger. This section describes how to create the required *Flash Method* (FME) file.

1. Select **Export...** from the **File** menu and then select **Flash Programmer** → **RealView Debugger Flash Device**, see Figure 6-18. Click on **Next**.

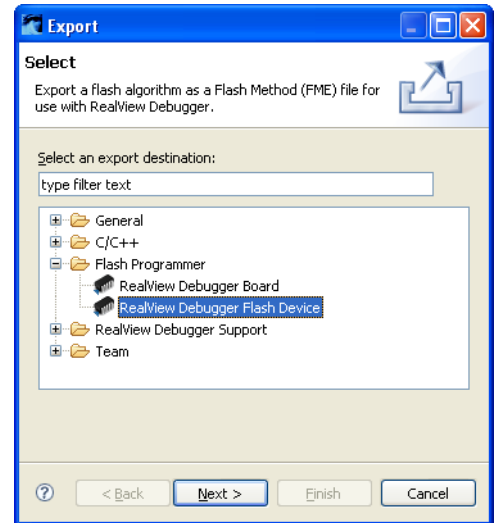


Figure 6-18 Export flash device to RealView Debugger

2. Select the flash device files to be exported. These can be files from an installed device or a project. Enter the name and location of the destination FME file or use **Browse...**, see Figure 6-19. Click on **Next**.

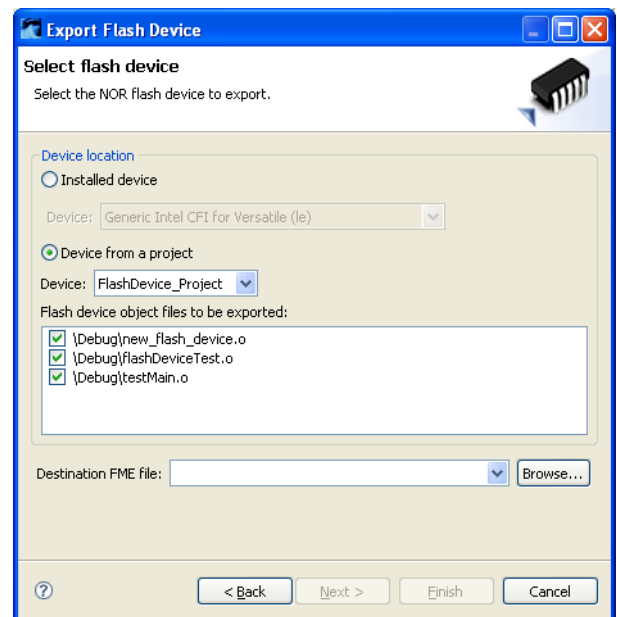


Figure 6-19 Export flash device object files

3. Enter the block structure for your flash device by either:
 - Using the **Add...** and **Edit...** buttons to enter the block structure for your flash device. Figure 6-20 and Figure 6-21 shows the device structure and block details. When the device structure is complete, click on **Next**.

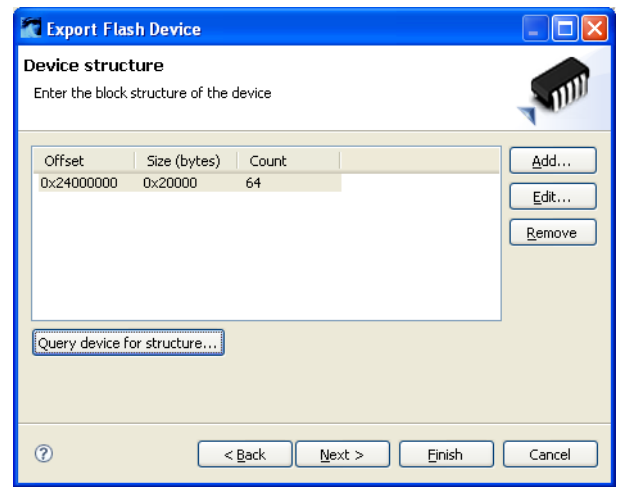


Figure 6-20 Export flash device structure

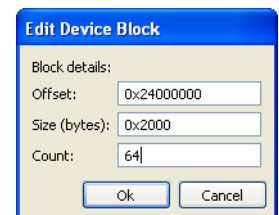


Figure 6-21 Export flash device block

- Alternatively, for installed devices, you can click on **Query device for structure...** to automatically set up the block structure for your flash device. Configure your target connection and click on **Query device...** to automatically read the block structure from the flash algorithm, see Figure 6-22. When the device structure is complete, click on **Next**.

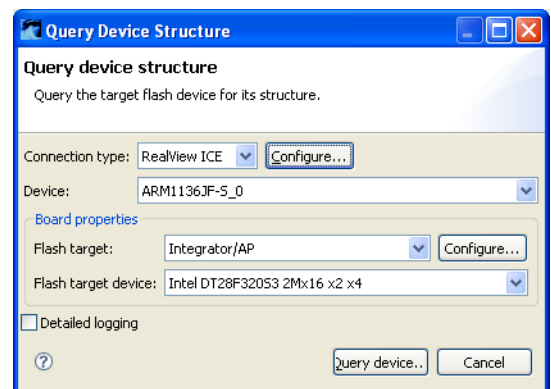
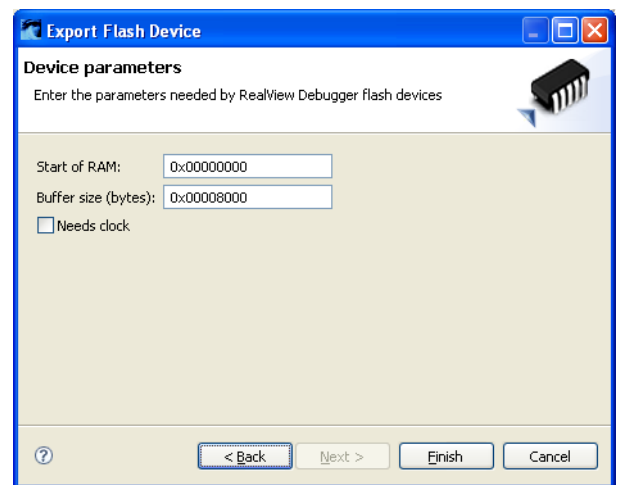


Figure 6-22 Query device structure

4. Figure 6-23 on page 6-17 shows the parameter dialog box. Enter the device parameters and click on **Finish** to save the FME file.

**Figure 6-23 Export flash device parameters**

Note

You must create a new *Board Chip Definition* (BCD) file to reference the exported FME file for your new flash device. See the *RealView® Debugger Target Configuration Guide* for more information.

Chapter 7

Working with RealView Debugger

This chapter describes how to launch RealView Debugger from the workbench, and how to configure the target connection settings for transferring to the debugger.

It includes the following:

- *Loading your executable image into RealView Debugger on page 7-2*
- *Creating your debug configuration on page 7-4*
- *Setting up your debug configuration on page 7-5*
- *Launching RealView Debugger using your debug configuration on page 7-7*
- *Exporting IP-XACT design files for use with RealView Debugger on page 7-8.*

7.1 Loading your executable image into RealView Debugger

The workbench enables target configurations to be set up and saved with your project files. These configuration files are stored in the `rvd` sub-folder within your project.

If your project contains target configuration files, the connection details are automatically transferred to RealView Debugger.

If your project does not contain target configuration files then you must manually set up a connection to your target.

See also:

- *Load using an existing workbench target configuration*
- *Load without a workbench target configuration.*

7.1.1 Load using an existing workbench target configuration

Ensure that RealView Debugger is closed before launching from the workbench to run or debug your executable image. When you launch RealView Debugger from the workbench the connection properties in RealView Debugger are automatically configured and all control passes to RealView Debugger. You must use the RealView Debugger interface to perform debug operations such as stepping, inserting breakpoints, and examining memory. For information on using RealView Debugger, see the *RealView® Debugger User Guide*. To load your executable image into RealView Debugger:

1. In the workbench, ensure your project is built and contains an executable image (see *Builds* on page 2-17).
2. Click on your executable image file in the Project Explorer view. Select **Run → Debug As → Load into RealView Debugger** from the main menu.
3. For a simulated target, the workbench launches RealView Debugger automatically and loads your executable image into it. If the target is not simulated then the connection properties in RealView Debugger are automatically configured but a manual connection and load is required.

The workbench remembers the last loaded executable image. If you want to reload the executable from the same project, press F11 on the keyboard. The project rebuilds if necessary, and the executable image is reloaded into RealView Debugger.

Note

If you want to build and debug a new target, you must close RealView Debugger before loading another executable image. This enables the new target configuration to be transferred from the workbench.

7.1.2 Load without a workbench target configuration

You can launch RealView Debugger from the workbench to run or debug your executable images. You must first create a connection to your target in RealView Debugger. When you launch RealView Debugger from the workbench, the image is loaded into the target and all control passes to RealView Debugger. You must use the RealView Debugger interface to perform debug operations such as stepping, inserting breakpoints, and examining memory. For information on using RealView Debugger, see the *RealView® Debugger User Guide*. To load your executable image into RealView Debugger:

1. Start RealView Debugger.

2. In RealView Debugger, create a connection to your target.
3. Close RealView Debugger.
4. In the workbench, ensure your project is built and contains an executable image (see *Builds* on page 2-17).
5. Click on your executable image file in the Project Explorer view. Select **Run** → **Debug As** → **Load into RealView Debugger** from the main menu.
6. The workbench launches RealView Debugger automatically and loads your executable image into it.

The workbench remembers the last loaded executable image. If you want to reload the executable from the same project, press F11 on the keyboard. The project rebuilds if necessary, and the executable image is reloaded into RealView Debugger.

7.2 Creating your debug configuration

You can create and set up your own debug configuration for each executable image in the workbench. To create a new debug configuration for your executable image:

1. Select your project from the Project Explorer view.
2. Select **Open Debug Dialog...** from the **Run** menu to show the Create, manage, and run configurations dialog box.
3. In the configurations panel on the left, select **RealView Debugger**.
4. Click on the toolbar or right-click and select **New** to create a new debug configuration for your project. The name of the executable image can be seen in the C/C++ Application field.

———— **Note** ————

If your project does not contain an executable image, or contains more than one executable image, then the C/C++ Application field remains blank. The workbench warns that the program does not exist and disables the **Debug** button on the panel. See *Selecting a different image to debug* on page 7-5, to set the executable image to debug.

5. A new debug configuration is created and has the project name by default. It is displayed under RealView Debugger in the Configurations panel.

7.3 Setting up your debug configuration

You can either create a new debug configuration or use an existing configuration to debug your executable image. If you have not already created a debug configuration, see *Creating your debug configuration* on page 7-4, to create a new configuration. This section describes how to set up an existing debug configuration.

See also:

- *Selecting an existing debug configuration*
- *Selecting a different image to debug*
- *Configuring RealView Debugger connection settings*
- *Specifying execution arguments* on page 7-6.

7.3.1 Selecting an existing debug configuration

To select an existing debug configuration:

1. Select **Open Debug Dialog...** from the **Run** menu to display the Create, manage, and run configurations dialog box.
2. Expand RealView Debugger in the Configurations panel.
3. Select the debug configuration you want to use.

The workbench provides several panels to set up or modify the selected RealView Debugger configuration. The **Main**, **Arguments**, and **Connection** tabs are described in the following sections.

7.3.2 Selecting a different image to debug

The **Main** tab provides options to associate a different project or executable image to the selected debug configuration:

Project You can either type the name of the project in the Project field, or click on **Browse...** and select from the list of available projects.

———— **Note** —————

You can only select a project that is currently open in the workbench.

C/C++ Application

You can either type the name of the image you want to debug, in the C/C++ Application field, or click on **Browse...** to select the executable image. Use **Search Project...** to show a list of executable images available to choose from in the current project. The executable images from different build configurations such as debug and release appear in this list.

7.3.3 Configuring RealView Debugger connection settings

Use the **Connection** tab to configure the RealView Debugger load options:

Connection

If RealView Debugger is connected to more than one target, you can select the target to load the image into, using the **Connections** drop-down list. Click on **Get Connections** for the workbench to obtain a list of available connections from RealView Debugger. Deselect the **Load into first target** to view the available

connections and select the one you require. If **Load into first target** is selected, the workbench loads the executable image into the target that is shown first in the list of available connections.

Parts to load

Use this to select the parts of the image to load into the target:

Symbols and Image

Use this to load all debug symbols and the program image.

Image Only

Use this to load the program image only, and not the debug symbols.

Symbols Only

Use this to load symbols only and not the program image.

Loading mode

Use this to select whether to replace the executable image already existing in the target:

Append Use this to append the new executable image to the existing image.

Replace Use this to replace the existing image with the image being loaded.

Sections

You can use the Sections field to specify the sections you want to load when the image is loaded. It is commonly used to reload the initialized data section when starting a program. Select **Load all sections** for the default option.

Set Program Counter (PC) to start address from object module

You can use this to set the PC to the start address specified in the ELF image, every time the image gets loaded into RealView Debugger.

7.3.4 Specifying execution arguments

You can specify arguments to the executable image in the **Arguments** tab:

Program arguments

You can specify a space-separated list of arguments to the executable, in the Program arguments field.

Variables

You can use variables supplied by the workbench, as arguments to your executable. Click on **Variables...** to select the ones you want or to create your own variables.

7.4 Launching RealView Debugger using your debug configuration

If you do not want to change the debug configuration for the executable you want to load into RealView Debugger, follow the steps in *Loading your executable image into RealView Debugger* on page 7-2.

If you do want to change any of the debug configuration settings before loading the executable into RealView Debugger, follow these steps:

1. Ensure that RealView Debugger is already connected to your target. See the *RealView® Debugger User Guide* for more information. Close any instance of RealView Debugger that is still running.
2. Select **Debug** from the **Run** menu.
3. Select your RealView Debugger configuration from the Configurations box or create a new debug configuration. See *Creating your debug configuration* on page 7-4.
4. If you want to change the project or image you want to debug, use the **Main** tab to select a different project or image.
5. If you want to modify or pass arguments to your executable image, use the **Arguments** tab. See *Specifying execution arguments* on page 7-6.
6. Select the **Connections** tab and then click on **Get Connections**. RealView Debugger launches and the workbench tries to connect to RealView Debugger.
7. When the workbench connects to RealView Debugger, it opens the **Connection** tab in the workbench Debug dialog box. If you have more than one target connection to RealView Debugger, deselect the **Load into first target** option. The available target connections are visible in the **Connection** drop-down list. Select the connection you want to use (see Figure 7-1). Modify the load settings as your require. See *Configuring RealView Debugger connection settings* on page 7-5.

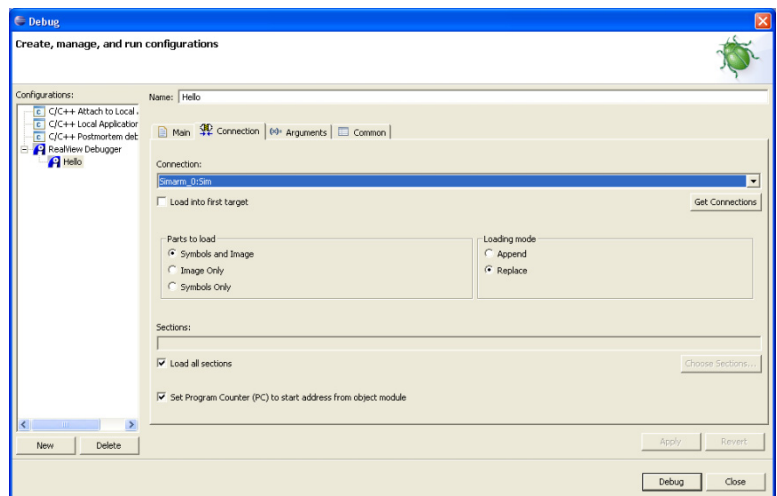


Figure 7-1 Debug panel

8. Click on **Debug**. The project rebuilds if necessary.
9. Your executable image is loaded into the target and you can debug it using RealView Debugger.

7.5 Exporting IP-XACT design files for use with RealView Debugger

Design files created with IP-XACT can be exported for use with RealView Debugger. This section describes how to create the required BCD file.

1. Select **Export...** from the **File** menu.
2. Select **RealView Debugger Support** → **IP-XACT to BCD**, see Figure 7-2. Click on **Next**.

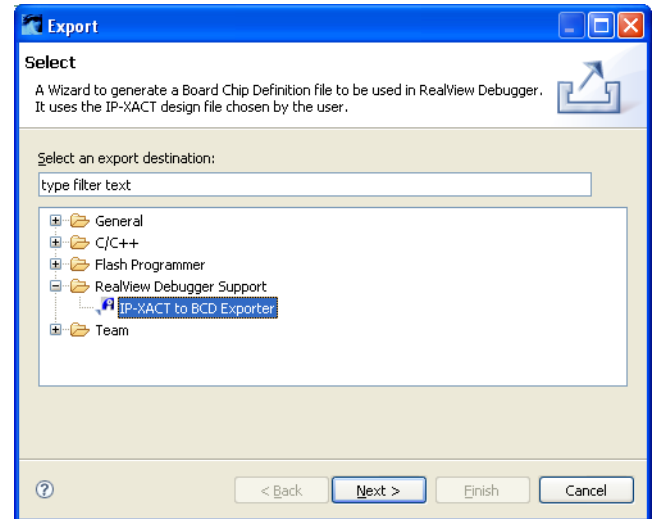


Figure 7-2 Export IP-XACT to RealView Debugger

3. Select the design file to be exported and enter the location of the destination BCD file or use **Browse...**, see Figure 7-3. Click on **Finish** to save the BCD file.

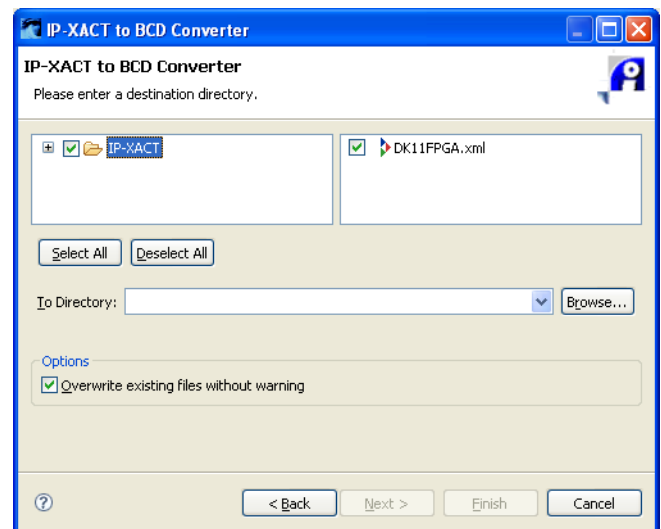


Figure 7-3 Select IP-XACT design file

Appendix A

Terminology, Shortcuts and Icons

This appendix describes some of the terminology used in the *ARM® Workbench IDE User Guide*. It also lists some of the most common keyboard shortcuts and menu or toolbar icons available for use when creating and building an ARM project.

It includes the following:

- *Terminology* on page A-2
- *Keyboard shortcuts* on page A-3
- *Menu and toolbar icons* on page A-5.

A.1 Terminology

This section describes some of the terminology used in the *ARM® Workbench IDE User Guide*. The list is in alphabetical order:

Block	A small sub-division of a flash device that can be programmed.
Dialog Box	A small page containing tabs, panels and editable fields prompting you to enter information.
Editor	A view that controls the visual aspects of source code for a specific file type.
Erase	A feature of a flash device where memory cells are reset to a known value.
Flash device	A set of flash memory that has a single command interface.
Panel	A small area in a dialog box or tab to group editable fields.
Perspective	A page within the workbench window containing a set of related views, editors, menus, and toolbars.
Program	A term used to describe the storing of data on a flash device.
Project	A group of related files and folders in the workbench.
Resource	A generic term used to describe a project, file, folder or a combination of these.
Send To	A term used to describe sending a file to a target.
Tab	A small overlay page containing panels and editable fields within a dialog box to group related information. Clicking on a tab brings it to the top.
Target	A location to send a file, for example, a flash image.
View	A small page to display related information for a specific function.
Width	The smallest number of bits (8, 16 or 32) that can be natively accessed by a flash device.
Wizard	A group of dialog boxes to guide you through a common tasks, for example, creating new files and projects.
Workbench	A window containing perspectives, menus, and toolbars.
Workspace	An area designated on your file system to store files and folders related to your projects.

A.2 Keyboard shortcuts

This sections lists some of the most common keyboard shortcuts available for use with the workbench:

F3 Click on an assembler label from a branch instruction or a C/C++ calling function and press F3 to move the editor focus to the position of the selected item.

F10 Use in conjunction with the arrow keys to access the main menu.

Alt+F4 Exit the workbench.

ALT+Left arrow

Go back in navigation history.

ALT+Right arrow

Go forward in navigation history.

Ctrl+; Provided with the ARM assembler editor to add comment markers to a selected block of code in the active file.

Ctrl+End Moves the editor focus to the end of the code.

Ctrl+Home Moves the editor focus to the beginning of the code.

Ctrl+F Opens the Find or Find/Replace dialog box to search through the code in the active editor. Some editors are read-only and therefore disable this functionality.

Ctrl+F4 Close the active file in the editor view.

Ctrl+F6 Cycles through open files in the editor view.

Ctrl+F7 Cycles through available views.

Ctrl+F8 Cycles through available perspectives.

Ctrl+F10 Use in conjunction with the arrow keys to access the drop-down menu.

Ctrl+L Move to a specified line in the active file.

Ctrl+Q Move to the last edited position in the active file.

Ctrl+Space Provides auto-completion on selected functions in editors.

Shift+F10 use in conjunction with the arrow keys to access the context menu.

Ctrl+Shift+F

Activates the code style settings in the Preferences dialog box and apply them to the active file.

Ctrl+Shift+L

Opens a small page with a list of all keyboard shortcuts.

Ctrl+Shift+R

Opens the Open resource dialog box.

Ctrl+Shift+T

Opens the Open Type dialog box.

Ctrl+Shift+/

Provided with the C/C++ editor to add comment markers to the start and end of a selected block of code in the active file.

A.3 Menu and toolbar icons

This section lists some of the most common menu and toolbar icons available for use with the workbench. For information on icons, markers, and buttons not listed in the following tables, use the standard *Workbench User Guide* or the *C/C++ Development User Guide* in the dynamic help.

Table A-1 ARM icons







Button	Description	Button	Description
ARM			
	Create new ARM project		Load into RealView Debugger
	Open flash programmer configuration dialog box		Open flash configuration dialog box
	Navigate to previous page		Navigate to next page

Table A-2 Perspective icons



Button	Description	Button	Description
	Open a new perspective		C/C++ perspective

Table A-3 View icons











Button	Description	Button	Description
	Open Overview		Open What's New
	Open Samples		Open Tutorials
	Open Workbench		Display drop-down menu
	Minimize view		Maximize view
	Restore view		Close view

Table A-4 Run and debug icons




Button	Description	Button	Description
	Run a program		Run an external tool
	Debug a program		

Table A-5 Editor icons









Button	Description	Button	Description
	Save the active file		Save all files
	Print the active file		Close file
	Create new configuration		Duplicate selected configuration
	Delete selected configuration		Collapse configuration tree

Table A-6 Editor markers








Button	Description	Button	Description
	Bookmark		Breakpoint marker
	Task marker		Search result
	Error marker		Warning marker
	Information marker		

Table A-7 Outline icons







Button	Description	Button	Description
	Hide fields		Hide static members
	Hide non-public members		Sort alphabetically
	Class		Namespace

Table A-7 Outline icons (continued)















Button	Description	Button	Description
#	Macro definition		Enum
	Enumerator		Variable
	Protected field		Private field
	Public field		Include
	Protected method		Private method
	Public method		Struct
	Type definition		Union
	Function		

Table A-8 Miscellaneous icons









Button	Description	Button	Description
	Open a new resource wizard		Open new project wizard
	Open new folder wizard		Open new file wizard
	open search dialog box		Display context-sensitive help
	Open import wizard		Open export wizard

Table A-9 Navigation icons
















Button	Description	Button	Description
	Go back		Go forwards
	Go back		Go forwards
	Open help instruction page		Synchronize TOC with active page
	Bookmark active page		Print active page

Table A-10 Help Contents icons

Button	Description	Button	Description
	Display list of all documents		Display list of documents in last search
	Display list of context-sensitive help links to related topics		Display list of all bookmarks
	Maximize frame		Restore frame
	Synchronize TOC with active page		