RealView® Development Suite

Version 3.1

Eclipse Plug-ins User Guide

ARM®

ARM DUI 0330D

RealView Development Suite
Eclipse Plug-ins User Guide

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
March 2006	A	Non-Confidential	Release for RVDS v3.0
March 2007	B	Non-Confidential	Release for RVDS v3.1
July 2007	C	Non-Confidential	Release for RVDS v3.1 - ARM flash programmer and assembler editor update
December 2007	D	Non-Confidential	Release for RVDS v3.1 - CodeWarrior Importer update

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Development Suite v3.1 Eclipse Plug-ins User Guide

Preface

This preface introduces the *RealView Development Suite Eclipse Plug-ins User Guide*. It contains the following sections:

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

About this book

This book provides user information on the Eclipse Plug-ins for *RealView Development Suite* (RVDS). It describes how to use the Eclipse *Integrated Development Environment* (IDE) to configure and build projects for ARM® targets. This book is not intended to familiarize the user with all aspects of the Eclipse IDE.

Intended audience

This book is written for all developers who are using the Eclipse IDE 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 ARM development tools. It does not assume that you are familiar with the Eclipse IDE.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the Eclipse Plug-ins for RealView Development Suite and how to access the online help.

Chapter 2 ***Creating an ARM Project***

Read this chapter for a tutorial on creating an ARM project.

Chapter 3 ***Importing an Existing Eclipse Project***

Read this chapter for information on importing existing Eclipse projects into the current Eclipse workspace.

Chapter 4 ***Importing a CodeWarrior Project***

Read this chapter for information on importing existing CodeWarrior® projects into the current Eclipse workspace.

Chapter 5 ***ARM Project Types***

Read this chapter for information about using different kinds of ARM project types provided within the Eclipse Plug-ins for RealView Development Suite.

Chapter 6 ***Configuring the Build and RealView Tools***

Read this chapter for information on how to configure the ARM RealView tools, to modify how your project is built.

Chapter 7 ***Working with the ARM Flash Programmer***

Read this chapter for information on how to use flash devices within Eclipse.

Chapter 8 ***Working with RealView Debugger***

Read this chapter for information on how to use RealView Debugger within Eclipse.

Appendix A ***Eclipse Terminology and Shortcuts***

Read this appendix for information on Eclipse terminology and shortcuts.

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

Typographical conventions

The following typographical conventions are used in this book:

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

Further reading

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

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

ARM publications

See the following publications for detailed documentation on various components of RVDS:

- *RealView Development Suite Getting Started Guide* (ARM DUI 0255)
- *RVCT Essentials Guide* (ARM DUI 0202)
- *RVCT Developer Guide* (ARM DUI 0203)
- *RVCT Assembler Guide* (ARM DUI 0204)
- *RVCT Compiler User Guide* (ARM DUI 0205)
- *RVCT Linker and Utilities Guide* (ARM DUI 0206)
- *RVCT Compiler Reference Guide* (ARM DUI 0348)
- *RVCT Libraries and Floating Point Support Guide* (ARM DUI 0349)
- *RVCT NEON Compiler Guide* (ARM DUI 0350)
- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Development Suite Glossary* (ARM DUI 0324).

Other publications

This book provides information specific to the Eclipse Plug-ins provided by ARM. For more information on the Eclipse IDE, visit the Eclipse web site at `http://www.eclipse.org`.

Feedback

ARM Limited welcomes feedback on both the Eclipse Plug-ins for RVDS, and its documentation.

Feedback on the Eclipse Plug-ins for RealView Development Suite

If you have any problems with the Eclipse Plug-ins for RVDS, please contact your supplier. To help them provide a rapid and useful response, please give:

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

Feedback on this book

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

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

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This book describes the Eclipse plug-ins for *RealView Development Suite* (RVDS), and how you can use it within Eclipse to develop software for ARM® targets. This book does not give detailed descriptions on all of the Eclipse features. For information on a specific feature, open the relevant dialog and access the F1 online help. See *Accessing online help* on page 1-6 for more information.

This chapter introduces the Eclipse plug-ins for RVDS and includes the following:

- *About the Eclipse plug-ins for RVDS* on page 1-2
- *Installation requirements* on page 1-4
- *Installing software updates for the Eclipse plug-ins for RVDS* on page 1-5
- *Accessing online help* on page 1-6.

1.1 About the Eclipse plug-ins for RVDS

The Eclipse plug-ins for RVDS integrate the following into the Eclipse *Integrated Development Environment* (IDE):

ARM RealView Compilation Tools

This plug-in enables software developers to use the Eclipse IDE as a project manager to create, build, debug, and manage C, C++ and assembly language projects for ARM targets. It also provides comprehensive configuration panels to specify options for the ARM compiler, assembler, linker, debugger, and other tools.

ARM Flash Programmer

This plug-in provides a new project wizard to create flash algorithms, configuration panels for sending images and managing target connections. It also provides a flash device export wizard for close integration with RealView Debugger.

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. The online help for this plug-in describes how to configure and activate these settings. See *Accessing online help* on page 1-6 for more information.

1.1.1 Restrictions of use

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

Organizing projects

The recommended structure for project source files is to create them in sub-directories below the project file. If a source file is created in a directory structure that is higher than the project, Eclipse creates an absolute link to that file.

Opening an existing Eclipse project

You must use the import wizard. See Chapter 3 *Importing an Existing Eclipse Project* for more information.

CodeWarrior sub-projects

CodeWarrior sub-projects are not supported. You must import each project individually. See Chapter 4 *Importing a CodeWarrior Project* 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 setup by referencing other projects that reside in your workspace. Select **Project** → **Properties** → **Project References** from the Eclipse main menu to manually add references.

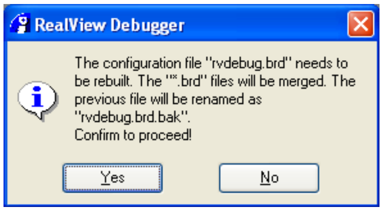
Link order Specifying the link order of your object files within the same project is not possible with Eclipse. As a workaround, if you split your object files into different projects, you can specify the project build order. Select **Windows** → **Preferences...** → **General** → **Workspace** → **Build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

Restore Defaults

Restoring the defaults of a project discards all information that is not a part of the project type. This means that any settings changed in the ARM New Project Wizard will be lost.

Starting RealView Debugger with an Eclipse target configuration

When you launch RealView Debugger from Eclipse using a project with an existing Eclipse target configuration, you might see the RealView Debugger dialog shown in Figure 1-1. Click **No** to use the target connection properties from Eclipse.


RealView Debugger

The configuration file "rvdebug.brd" needs to be rebuilt. The "*.brd" files will be merged. The previous file will be renamed as "rvdebug.brd.bak".
Confirm to proceed!

Figure 1-1 RealView Debugger message

1.2 Installation requirements

This section give an overview of the components required to use Eclipse.

1.2.1 RVDS installation

RVDS Install RVDS v3.1 from CD and all the required components are installed for you, including Eclipse.

1.2.2 Custom installation

If you have a custom installation of Eclipse then ensure that the following components are installed before using the Eclipse plug-ins for RVDS.

Eclipse Download and install Eclipse v3.2 from `http://www.eclipse.org`.

Eclipse plug-ins for RVDS

Use the Software Updates feature of Eclipse to install the Eclipse plug-ins for RVDS, from `http://www.arm.com/eclipse`. See *Installing software updates for the Eclipse plug-ins for RVDS* on page 1-5.

JRE Download and install the version of Java Runtime Environment (JRE) that works best on your platform. See `http://www.java.com` or `http://www.eclipse.org`.

CDT The C and C++ Development Tool (CDT) is a plug-in that integrates the C and C++ build environment into the Eclipse IDE. This must be installed because the RealView tools plug into the CDT. CDT v3.1 is required. Install the latest CDT from `http://www.eclipse.org`.

GNU make GNU make v3.80 is required. Install MinGW from `http://www.mingw.org` or Cygwin from `http://www.cygwin.com`. MinGW is recommended.

1.3 Installing software updates for the Eclipse plug-ins for RVDS

If you installed RVDS v3.1 from CD then all the required Eclipse components are already installed.

To install updated features of the Eclipse plug-ins for RVDS you must use the Software Updates feature in Eclipse. The first time you use the Software Updates feature, you must create an update site to download the Eclipse plug-ins:

1. Launch the Eclipse IDE.
2. Select **Help** → **Software Updates** → **Find and Install...**
3. From the **Install/Update** dialog, select **Search for new features to install**. Click **Next**.
4. You must create a new site to locate the Eclipse plug-ins. In the **Install** dialog, click **New Remote Site...**
5. In the **New Update Site** dialog, enter any name, for example `Eclipse plug-ins for RVDS`. In the URL text box enter `http://www.arm.com/eclipse` and click **OK**.
6. Eclipse automatically selects your newly created update site from the list of **Sites to include in search**. Click **Finish**.
7. In the **Updates** dialog, select your Eclipse plug-ins for RVDS update site, and click **Next**.
8. Read the ARM license agreement and accept it. If you do not accept the license agreement, you cannot install the Eclipse plug-ins. Click **Next**, and then click **Finish**.
9. In the Verification dialog, click **Install All**. Eclipse installs the Eclipse plug-ins for RVDS.
10. Eclipse requests your permission to restart. Click **Yes** to restart Eclipse. The latest Eclipse plug-ins for RVDS has been installed.

To update the Eclipse plug-ins, in the future, follow the same steps as for installing the Eclipse plug-ins, without creating a new remote site. The same procedure can be used to install other plug-ins.

——— **Note** ———

You can enable automatic updates in the **Install/Updates** panel of the **Preferences** dialog, select **Window** → **Preferences...** from the Eclipse main menu to access it.

———————

1.4 Accessing online help

To access the F1 online help for the ARM RealView tools and related options, click on an editable field for the feature that you are interested in and press F1 on your keyboard.

An alternative way to get online help is:

1. Select **Help** → **Help Contents** from the Eclipse main menu.
2. From the **Contents** on the left, select **RealView Development Suite Online Help** and then select the plug-in tool that you want help on.

Chapter 2
Creating an ARM Project

You can use Eclipse to create RVDS projects for ARM targets. Eclipse generates appropriate makefiles for your RVDS projects. You can also create Managed Make and Standard Make projects, see the *C/C++ Development User Guide* in Eclipse Help for more information.

This chapter describes how to create a new RVDS project for ARM. It includes the following:

2.1 Creating an RVDS project for ARM

You can create a new ARM project using any of the project types provided with RVDS. This section describes how to create a new Executable (ARM) project:

1. Select **File** → **New** → **RVDS Project for ARM** from the Eclipse main menu, see Figure 2-1.

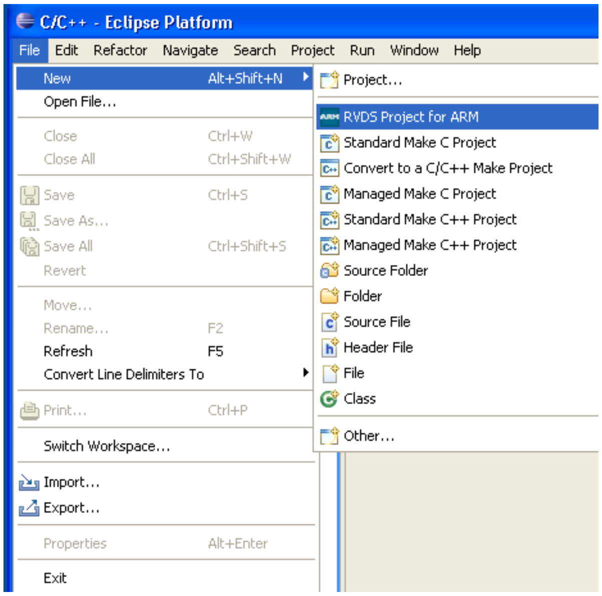


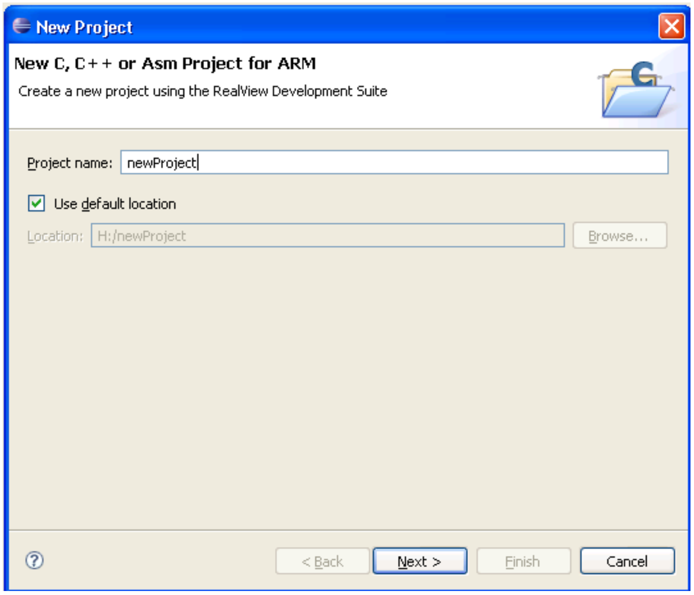
Figure 2-1 Creating a new ARM project

——— **Note** ———

RVDS Project for ARM combine the features of a **Managed Make C Project** and a **Managed Make C++ Project**. You can have `.c`, `.cpp` and `.s` files within the same project.

———————————

2. Enter a name for your project, see Figure 2-2 on page 2-3.


New Project

New C, C++ or Asm Project for ARM

Create a new project using the RealView Development Suite

Project name: newProject

☑ Use default location

Location: H:/newProject

Figure 2-2 Naming your project

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

4. Select **Project Type**, see Figure 2-3 on page 2-4.

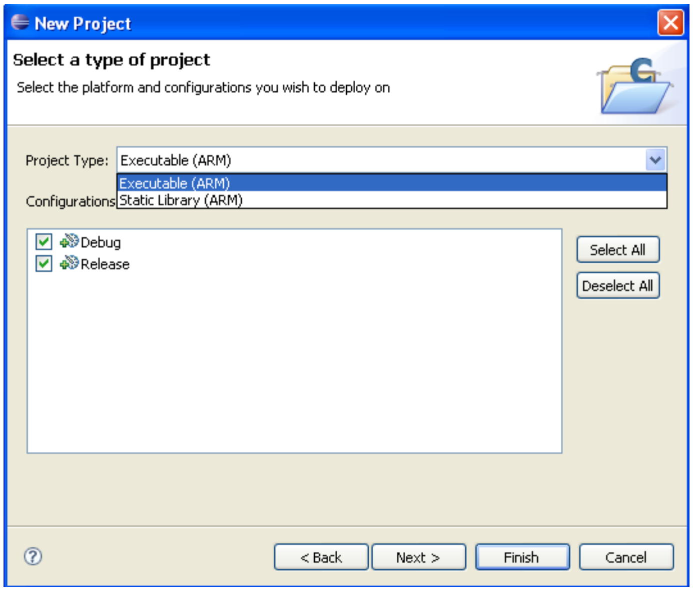


Figure 2-3 Choosing your project type

5. In the **Configurations** panel, leave the **Debug** and **Release** options selected to enable you to change the project settings for different configurations. Click **Next**.

6. Figure 2-4 on page 2-5 shows the **Additional Project Settings** panel. If you have inter-dependent projects, select any that apply to this project and click **Next**.

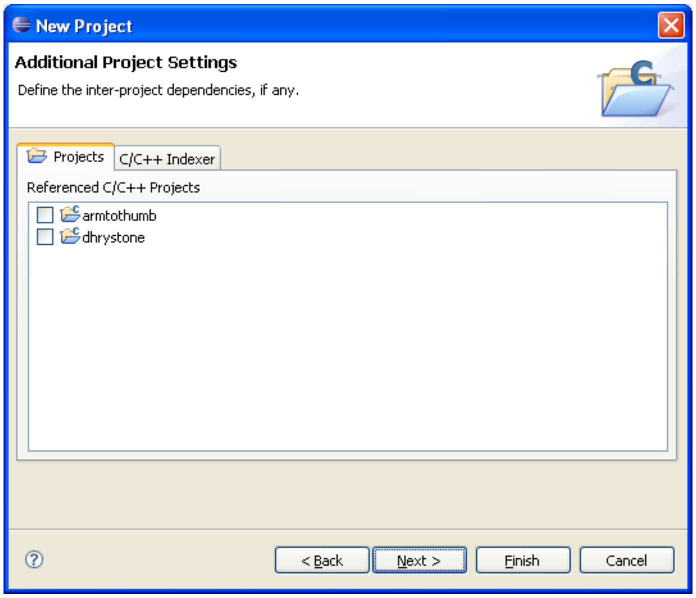

New Project
Additional Project Settings
Define the inter-project dependencies, if any.
Projects
C/C++ Indexer
Referenced C/C++ Projects
armtothumb
dhrystone
< Back
Next >
Finish
Cancel

Figure 2-4 Additional project settings

7. Figure 2-5 shows a typical **Target Settings** panel. Select **Target** if required. Click **Next**.

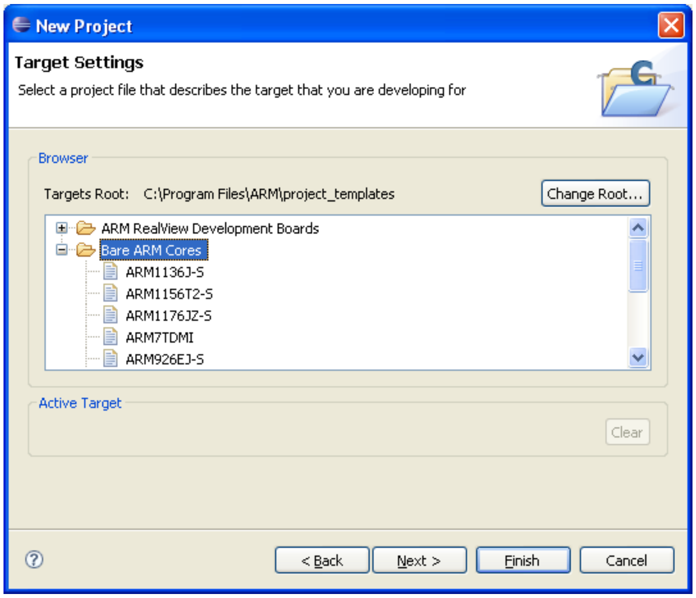

New Project
Target Settings
Select a project file that describes the target that you are developing for
Browser
Targets Root: C:\Program Files\ARM\project_templates
Change Root...
ARM RealView Development Boards
Bare ARM Cores
ARM1136J-S
ARM1156T2-S
ARM1176JZ-S
ARM7TDMI
ARM926EJ-S
Active Target
Clear
< Back
Next >
Finish
Cancel

Figure 2-5 Typical target Settings

Note

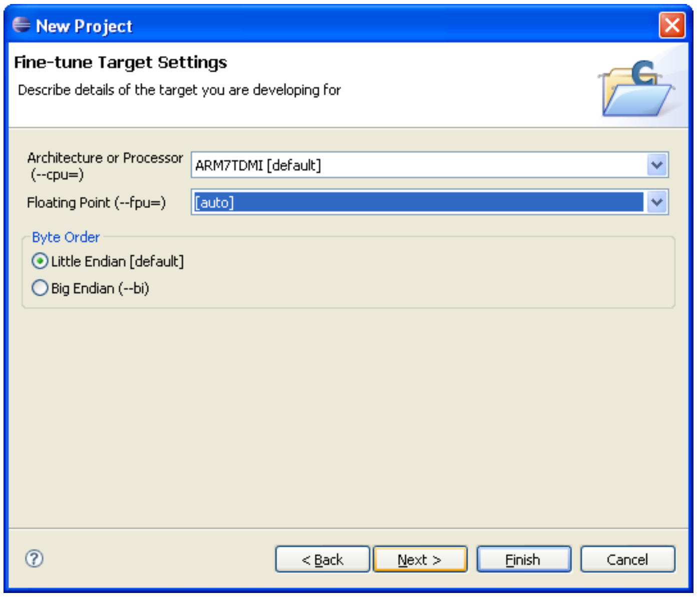
Selecting a target in the **Target Settings** panel installs associated information for use by the RealView Debugger. This information is transferred when you launch RealView Debugger from Eclipse to run or debug your executable images.

8. Figure 2-6 shows the **Fine-tune Target Settings** panel. Select **Architecture or Processor (--cpu=)**, **Floating Point (--fpu=)** and **Byte Order**.

   Note

   Selecting a VFP target here, requires initialization code and possible support code to use VFP in your project. For more information see the *Application Note 133 Using VFP with RVDS* or try the VFP examples provided with RVDS in the main examples directory: `*install_directory*\RVDS\Examples\...\vfpsupport`.

   Click **Next**.



New Project

Fine-tune Target Settings

Describe details of the target you are developing for

Architecture or Processor (--cpu=): ARM7TDMI [default]

Floating Point (--fpu=): [auto]

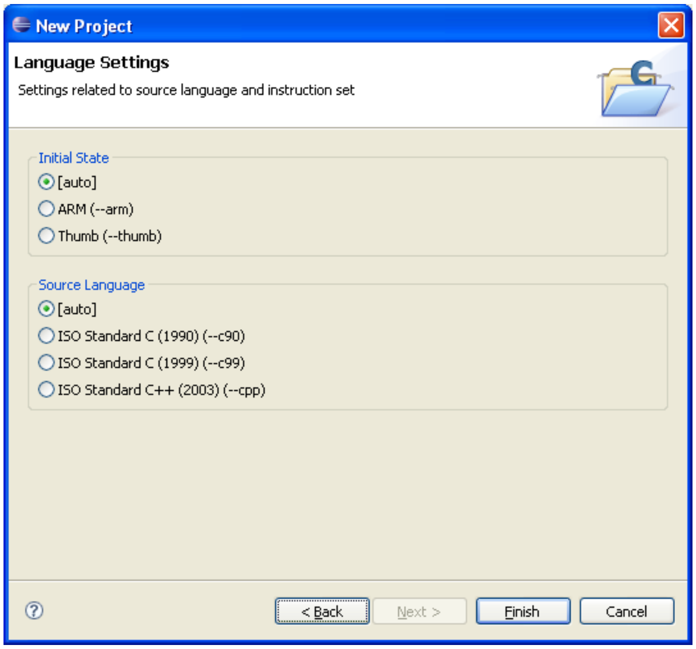
Byte Order

(●) Little Endian [default]

( ) Big Endian (--bi)

Figure 2-6 Fine-tune target settings

9. Figure 2-7 on page 2-7 shows the **Language Settings** panel. Select the **Initial State** and **Source Language** if required. The default setting **[auto]** enables the compiler to select the best options for your project. Click **Finish** to create your project.



New Project

Language Settings

Settings related to source language and instruction set

Initial State

[✓ [auto]]
ARM (--arm)
Thumb (--thumb)

Source Language

[✓ [auto]]
ISO Standard C (1990) (--c90)
ISO Standard C (1999) (--c99)
ISO Standard C++ (2003) (--cpp)

Figure 2-7 Language settings

10. The new project is visible in the **C/C++ Projects** view.

2.2 Adding a new file to the project

To add an empty source file to the newly created project:

1. Right-click on your project in the **C/C++ Projects** view.
2. Select **New** → **File** to display the **New file** dialog, see Figure 2-8.

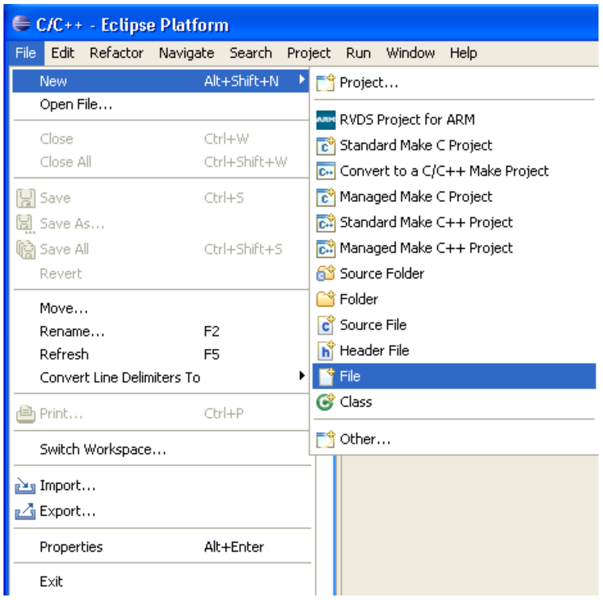


Figure 2-8 Adding a new source file

3. Select your project in the **Enter or select the parent folder** text box, see Figure 2-9 on page 2-9.
4. Enter a name and extension for the source file in the **File name** text box.

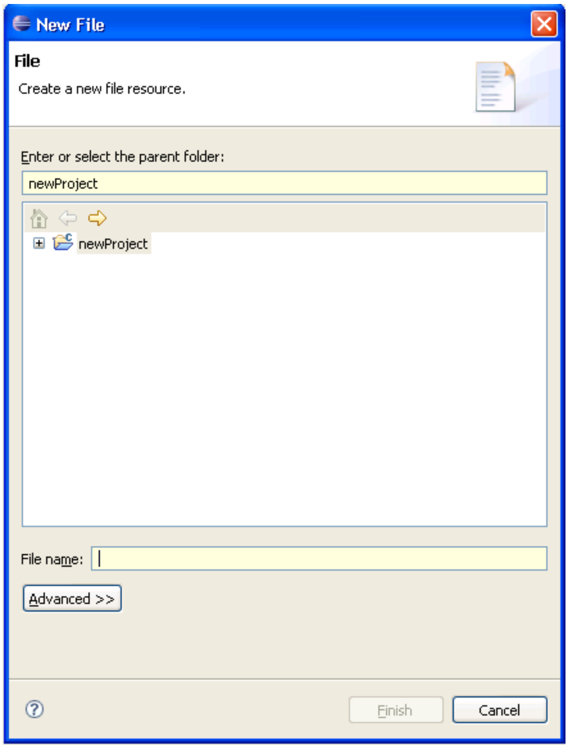


Figure 2-9 Naming your file

5. Click **Finish**. The source file is displayed in the **C/C++ Projects** view.

Note

- You can also create files, or drag and drop files directly into the project directory, using the file explorer. To view these files in Eclipse, right click the project in the **C/C++ Projects** view and select **Refresh**.
- You can also drag and drop files directly into the project directory, in the **C/C++ Projects** view in Eclipse.

2.3 Building the project

Once you have added files to your project, the project is built for you if **Project → Build Automatically** on the Eclipse main menu is selected. Or you can manually build your project by:

1. Selecting the project directory from the **C/C++ Projects** view.
2. Selecting **Project → Build Project** from the Eclipse main menu. Alternatively, you can right-click the project in the **C/C++ Projects** view and select **Build Project** from the context menu.

   The build output has the name of the project by default. A sub-directory gets created in the project directory for the active build configuration, and the build output is saved in this sub-directory. By default, the active build configuration is called `Debug`. See *About the predefined configurations* on page 5-4.

——— **Note** ———

By default Eclipse is configured to perform builds automatically when you save files. To disable this feature, ensure that **Project → Build Automatically** is not selected.

Be aware that files are not saved automatically when you do a manual build!

————————

Chapter 3
Importing an Existing Eclipse Project

The Eclipse environment uses the import process to select and build existing projects within the Eclipse workspace. This chapter describes the import process for projects. It includes the following:

- *Selecting and building an existing Eclipse project* on page 3-2.

3.1 Selecting and building an existing Eclipse project

This section describes how to import an existing Eclipse project into the Eclipse workspace.

1. Select **File** → **Import...** from the Eclipse main menu, see Figure 3-1.

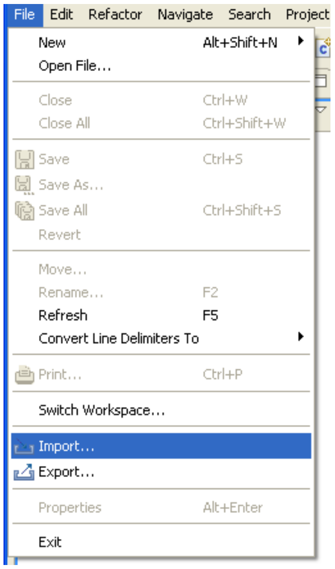
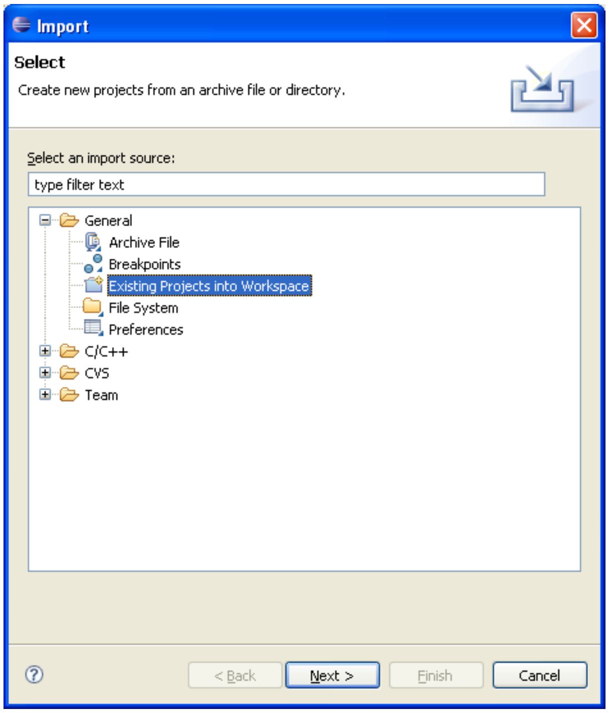


Figure 3-1 Import option within Eclipse

2. To import an existing project, select **Existing Project into Workspace**. Click **Next**. See Figure 3-2 on page 3-3.


Import

Select

Create new projects from an archive file or directory.

Select an import source:

type filter text

Figure 3-2 Selecting the import source type

3. Click on **Browse** to select the project root directory. 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 3-3 on page 3-4.

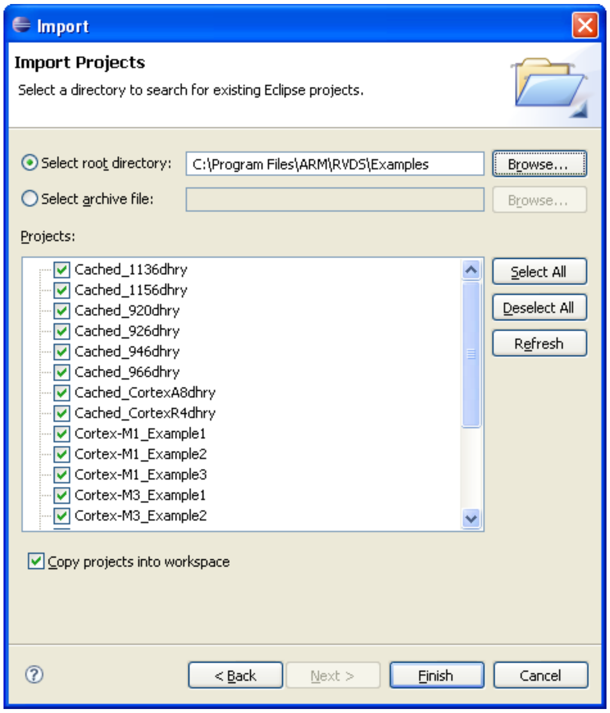


Figure 3-3 Selecting existing Eclipse projects for import

4. Click **Finish**. See Figure 3-3.

——— **Note** ———

If **Build Automatically** is selected, the project will build when you click **Finish**. If not, you will need to select **Project** → **Build Project** from the Eclipse main menu to complete the build step.

———————

Chapter 4
Importing a CodeWarrior Project

This chapter describes how to import existing CodeWarrior® ARM projects into the Eclipse IDE. It includes the following:

4.1 About the CodeWarrior importer

The Eclipse Plug-ins for RVDS enable you to import projects created using CodeWarrior for RVDS, into the Eclipse environment. In order to do so, you must first 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 directory as the CodeWarrior project file (with the `.mcp` filename extension)

———————

Refer to the *RealView Development Suite CodeWarrior IDE Guide* for more information on how to export your CodeWarrior projects into XML. When you import CodeWarrior projects in Eclipse, a new Eclipse Managed Make project is created. The ARM RealView tool settings in the CodeWarrior project are applied to the new Eclipse project.

4.1.1 Limitations

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 new file to the project* on page 2-8 for more information.
- Command-line options that do not map to panel settings in Eclipse are included in the **Tool Settings** panel as **Extra** → **Extra options**. See *Accessing the build properties for an ARM project* on page 6-2 for more information.

4.2 Importing a CodeWarrior project

To import a CodeWarrior project into your workspace:

1. Select **File** → **Import** from the main menu.

2. Select **CodeWarrior Project exported as XML** from the **Import** dialog, see Figure 4-1. Click **Next**.

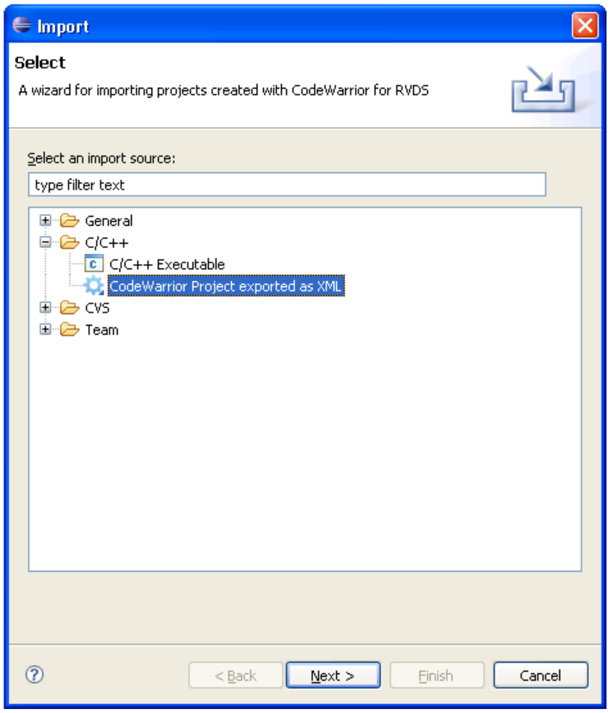
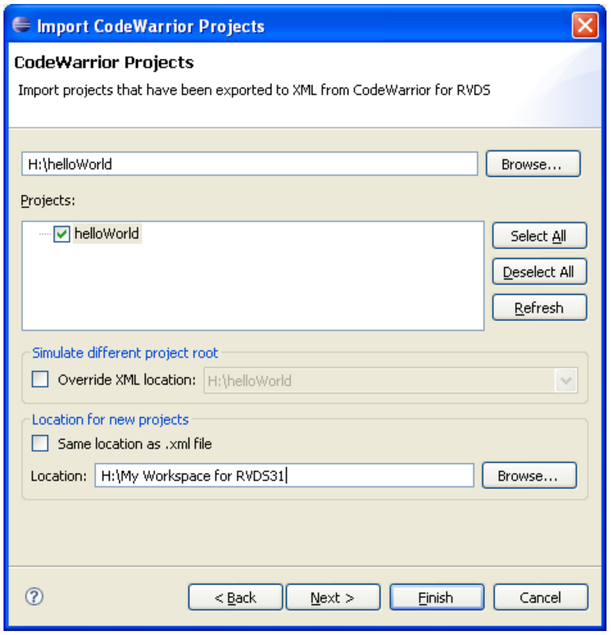


Figure 4-1 Selecting CodeWarrior XML

3. Use **Browse...** to select the folder containing the `.xml` file, see Figure 4-2 on page 4-4.



CodeWarrior Projects

Import projects that have been exported to XML from CodeWarrior for RVDS

H:\helloWorld

Projects:

[✓] helloWorld

Simulate different project root

[ ] Override XML location: H:\helloWorld

Location for new projects

[ ] Same location as .xml file

Location: H:\My Workspace for RVDS31

Figure 4-2 Importing a CodeWarrior Project

4. 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.

5. If your source files are located in a folder above the CodeWarrior project root, you can use the **Override XML location** option, see Figure 4-3. By selecting a different project root, the import wizard can mirror the parent structure when creating the new project. Figure 4-3 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.

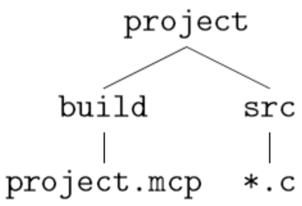


Figure 4-3 Structure with the source folder above the project file

6. In the **Location** text box, use 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.

   ————————

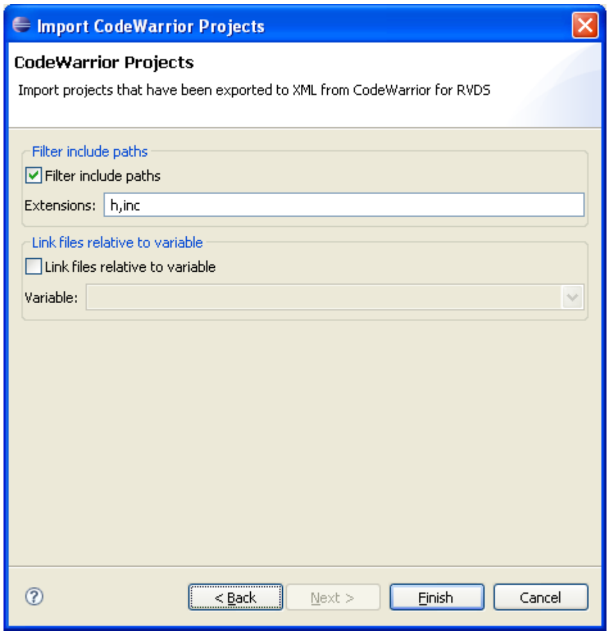
7. Use the **Filter include paths** option to minimize the include paths option on the command-line. Only paths containing files ending with the specified extensions are added to the command-line options. Extensions must be given as a comma-separated list. For example: *h,inc*.

   ——— **Note** ———

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

   ————————

8. The location of linked resources can be specified relative to a path variable. 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, see Figure 4-4. Click on **Finish** to create your new project.


Import CodeWarrior Projects

CodeWarrior Projects

Import projects that have been exported to XML from CodeWarrior for RVDS

Filter include paths

[✓] Filter include paths

Extensions: h,inc

Link files relative to variable

[ ] Link files relative to variable

Variable:

Figure 4-4 Filter include paths and path variables

Note

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

When you import a project using a path variable, the wizard automatically updates the **C/C++** → **PathEntry Variables** option in the **Preferences** dialog to correspond with the **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 **C/C++ Projects** view.

Note

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

Chapter 5
ARM Project Types

This chapter provides information on the different ARM project types provided by the Eclipse Plug-ins for RVDS. It includes the following:

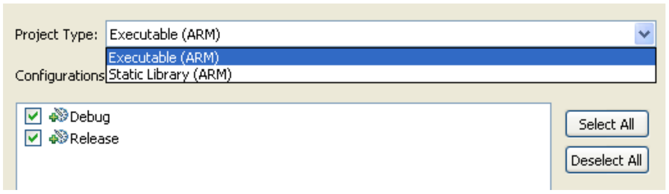
- *About the ARM project types* on page 5-2
- *About the predefined configurations* on page 5-4.

5.1 About the ARM project types

The Eclipse Plug-ins for RVDS provides the following ARM project types:

- executable
- static library.

You can select a project type from the drop-down box when creating a new managed make project, see Figure 5-1. The ARM project types are described in the following sections.



Project Type: Executable (ARM)

Configurations:

- Executable (ARM)
- Static Library (ARM)

Figure 5-1 ARM project types

5.1.1 Executable

Use the **Executable** project type to create an executable ELF image from ARM or 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 Eclipse to call the ARM `fromelf` utility to create a binary file from the executable ELF image. See *Using the ARM fromelf utility* on page 6-6.

5.1.2 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 major differences are:

- 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.

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

5.2 About the predefined configurations

Any project created using an ARM project type 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 ARM project types the `Debug` configuration is set active by default. The active configuration can be changed in the **C/C++ Build** configuration panel. See Chapter 6 *Configuring the Build and RealView Tools* for information on modifying the tool options.

Chapter 6

Configuring the Build and RealView Tools

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

——— **Note** ———

Build configuration settings can be applied to individual files and complete projects. To access the **Properties** dialog, right-click on either a project folder or a file in the **C/C++ Projects** view.

———————

It includes the following:

6.1 Accessing the build properties for an ARM project

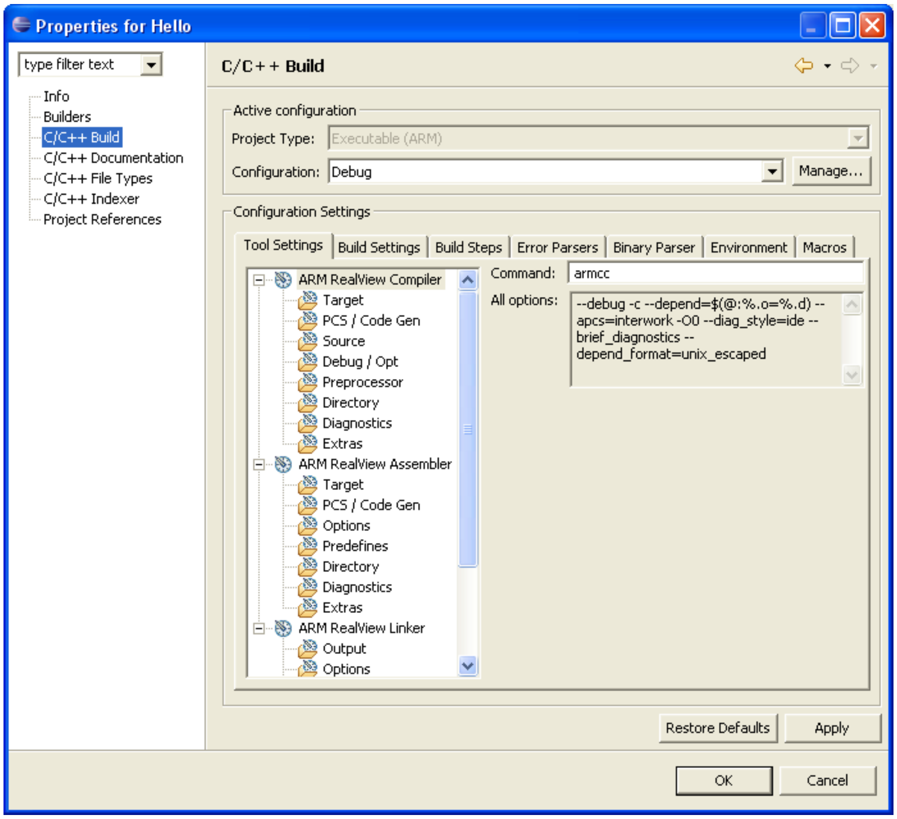
The **C/C++ Build** configuration panels enable you to setup the RealView tools for a specific build configuration, in your project. This section describes how to access the build configuration panels that affect all the source files in your project:

1. Select a project in the **C/C++ Projects** view. If there is no project available, add a project to your workspace by either:
   - importing an existing ARM project, see Chapter 3 *Importing an Existing Eclipse Project*
   - creating a new project using an ARM project type, see *Creating an RVDS project for ARM* on page 2-2.
2. Select **Project** → **Properties** from the Eclipse main menu. You can also right-click on a project in the **C/C++ Projects** view and select **Properties** from the context menu.
3. Select **C/C++ Build** from the **Properties** dialog.
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 RealView tool settings for the release build.
   - Use the **Configuration** drop down menu to select **Debug**. This shows the RealView 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.

5. The RealView tools available for the project, and their respective configuration panels are displayed in the **Tool Settings** tab of the **Configuration Settings** panel (Figure 6-1 on page 6-3).



Project Type: Executable (ARM)
Configuration: Debug

Command: armcc

All options:
```
--debug -c --depend=$(@:%.o=%.d) --
apcs=interwork -O0 --diag_style=ide --
brief_diagnostics --
depend_format=unix_escaped
```

Figure 6-1 Build configuration panel for an ARM project

For more information, see:

- *Configuring ARM RealView tools* on page 6-5
- *Using the ARM fromelf utility* on page 6-6.

6.2 Accessing the build properties for a specific file

The RealView compiler and RealView assembler can be setup differently for each source file. If you specify different tool options for a source file, it overrides the options specified in the project configuration panels that apply to all source files. See *Accessing the build properties for an ARM project* on page 6-2 for information. This section describes how to access the configuration panel for any specific source file in your project:

1. Select a project in the **C/C++ Projects** view. If there is no project available, add a project to your workspace by either:
    - importing an existing ARM project, see Chapter 3 *Importing an Existing Eclipse Project*
    - creating a new project using an ARM project type, see *Creating an RVDS project for ARM* on page 2-2.
2. If there are no source files in your project, add a new source file (with `.c`, `.cpp`, or `.s` extension) to your project. See *Adding a new file to the project* on page 2-8.
3. Right-click on a source file in your project and select **Properties** from the context menu.
4. Select **C/C++ Build** from the **Properties** dialog.
5. This displays the configuration panel specific to the selected file. The RealView tool that you can configure is shown in the **Tool Settings** tab. If your source file has a `.s` extension, the **Tool Settings** tab shows the **ARM RealView Assembler** settings. If your source file has a `.c` or `.cpp` extension, the **Tool Settings** tab shows the **ARM RealView Compiler** settings.

For more information, see:

- *Configuring ARM RealView tools* on page 6-5
- *Using the ARM fromelf utility* on page 6-6.

6.3 Configuring ARM RealView tools

To enable you to view and set the options easily, a number of panels are provided for each of the ARM RealView tools 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 6-2.

——— **Note** ———

- The **ARM RealView Linker** panels are only displayed for executables.
- The **ARM RealView Librarian** panels are only displayed for static library types.

See Chapter 5 *ARM Project Types* for more information.

———————

For a description of each option shown in these panels, access the F1 online help. See *Accessing online help* on page 1-6 for more information. The parent panel of each tool contains an **All options** text box that shows all the options that you set for that tool. This list of options is passed to the RealView tool when it is invoked.

——— **Note** ———

- The **All options** text box contains additional options to enable the RealView tool to work correctly in the Eclipse environment.
- Options that are set to their RealView tool defaults, do not need to get passed to the tool when it is invoked, and thus are not included in the **All options** text box. For example, if the compiler optimization level is set to the default (level 2), the `-O2` option does not appear in the **All options** text box.

———————

Each tool has an **Extras** panel 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.

6.4 Using the ARM fromelf utility

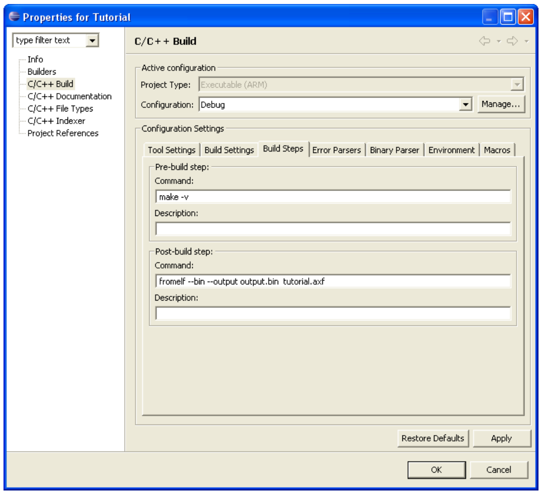
This section contains:

- *Converting ELF images to other formats*
- *Disassembling code* on page 6-7.

6.4.1 Converting ELF images to other formats

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 the *RealView Compilation Tools Linker and Utilities Guide* for more information on using `fromelf`, including information on output formats. To configure Eclipse 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 6-2.
2. Click the **Build Steps** tab in the **C/C++ Build** panel.
3. In the **Command** text box of the **Post-build step**, enter `fromelf --bin --output output.bin` `*inputfile*.axf`, where *inputfile* is the name of the executable ELF image. See Figure 6-2.
4. The binary file gets created when you build the project. The binary file is saved as `output.bin` in the sub-directory for the active build configuration, for example `Debug`.



Active configuration

Project Type: Executable (ARM)

Configuration: Debug

Configuration Settings

Pre-build step:

Command: make -v

Description:

Post-build step:

Command: fromelf --bin --output output.bin tutorial.axf

Description:

Figure 6-2 Calling `fromelf` from Eclipse

6.4.2 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 **C/C++ Projects** view to show the object files.
2. Right-click on the object file you want to disassemble, and select **OpenWith** → **FromElfOpener**. Alternatively, you can simply double-click the object file.
3. The disassembled object file is displayed in the Eclipse file editor.
4. You can save the disassembled file.

6.5 Restoring defaults

You can use **Restore Defaults** on the configuration panel (see *Accessing the build properties for an ARM project* on page 6-2) to reset the settings of all the RealView 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 **Restore Defaults** on the project configuration panel (see *Accessing the build properties for an ARM project* on page 6-2), 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 6-4).
- If you click **Restore Defaults** on 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 7
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:

7.1 About the ARM flash programmer

The ARM flash programmer connects to your target using 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.

You can use the ARM flash programmer plug-in to:

- Send an image to a specific memory location on your flash device. See *Programming a flash device* on page 7-4 for more information.
- Create a new flash algorithm for use in an Eclipse project or for export and distribution to a 3rd party. See *Creating a new flash algorithm* on page 7-12 for more information.

The flowchart in Figure 7-1 on page 7-3 illustrates both processes.

In order to send an image to your flash device, you must setup the following:

- flash image, Table 7-1 lists the supported image types
- flash programmer target configuration
- RealView ICE.

Table 7-1 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

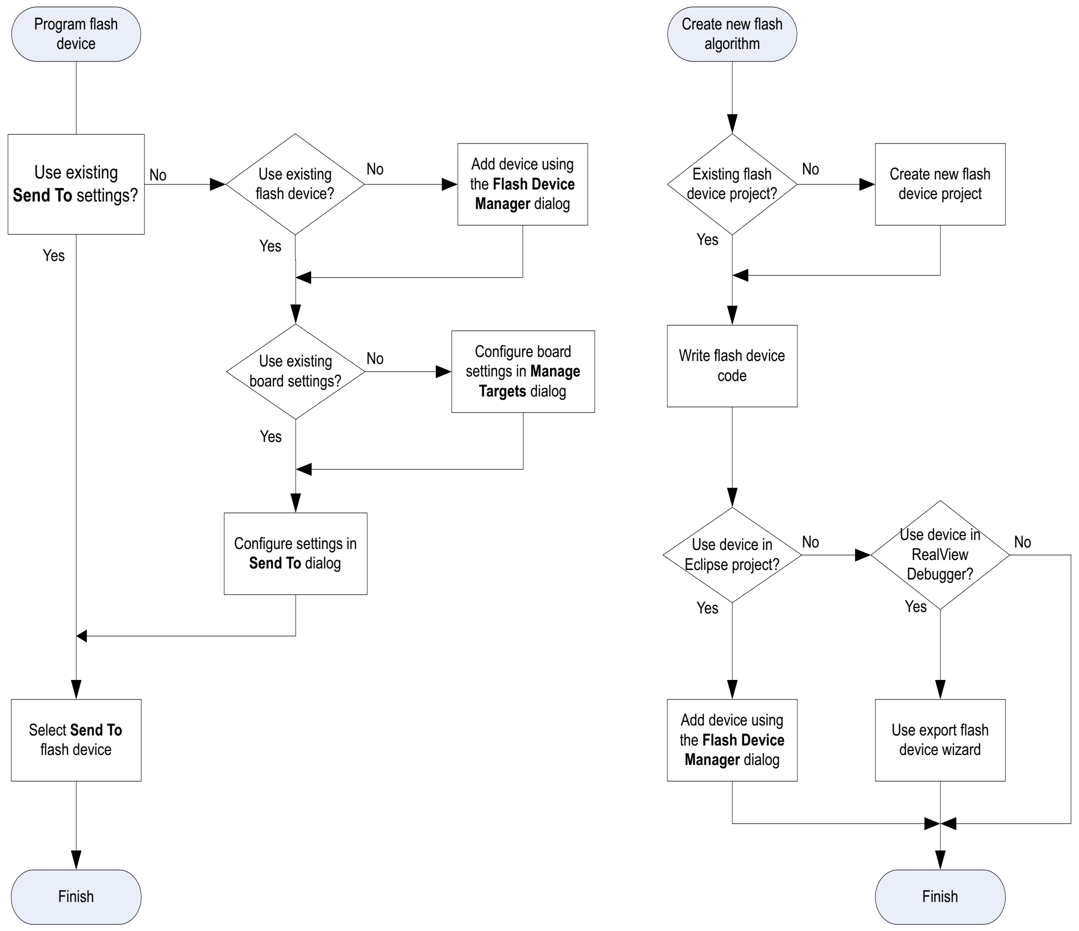

Program flash device
Use existing **Send To** settings?
No
Yes
Use existing flash device?
No
Yes
Add device using the **Flash Device Manager** dialog
Use existing board settings?
No
Yes
Configure board settings in **Manage Targets** dialog
Configure settings in **Send To** dialog
Select **Send To** flash device
Finish
Create new flash algorithm
Existing flash device project?
No
Yes
Create new flash device project
Write flash device code
Use device in Eclipse project?
No
Yes
Use device in RealView Debugger?
No
Yes
Add device using the **Flash Device Manager** dialog
Use export flash device wizard
Finish

Figure 7-1 The ARM flash programmer

7.2 Programming a flash device

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

- If you have previously setup a flash programmer configuration for your device, select **Run** → **Send To** → **Send To** → ***Flash Image*** from the Eclipse main menu.
- If your flash programmer configuration is not setup:
    1. Click **Run** → **Send To** → **Send To**... from the Eclipse main menu.
    2. Use the tabs in the **Send To** dialog to complete this process, see Figure 7-2 on page 7-5.

       Image

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

       Connection

       Use to setup the connection method for your flash device. The CPU option is populated for you when you configure your connection method.

       Flash device

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

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

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

       Click **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 F1 online help. See *Accessing online help* on page 1-6 for more information.

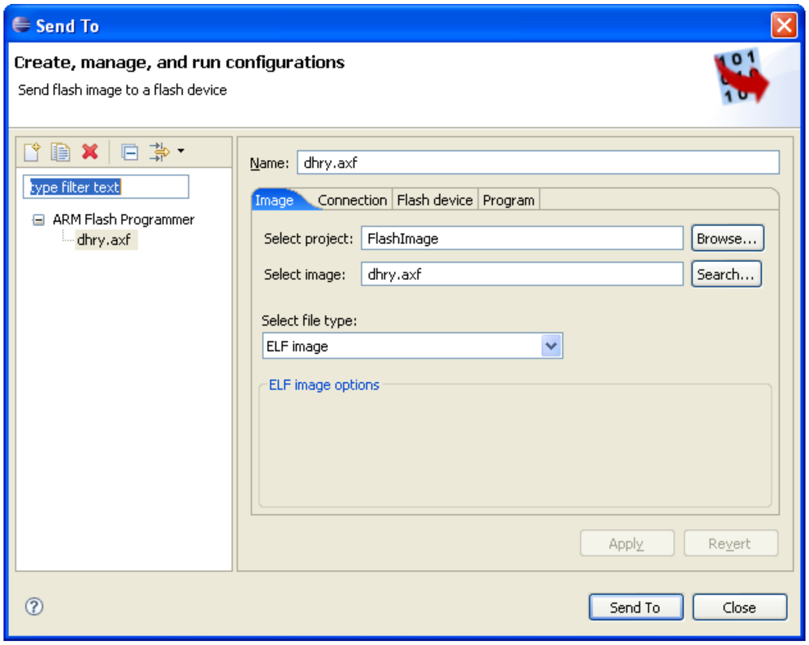


Figure 7-2 Create, manage, and run configurations

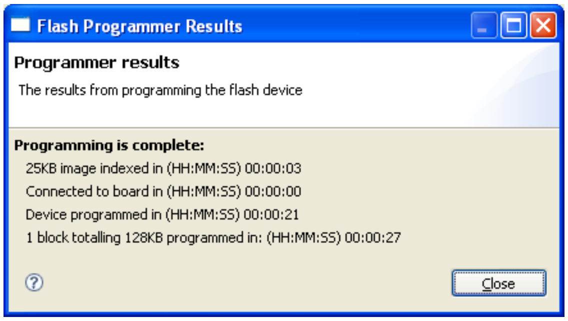
3. Click **Send To** to send your flash image to your flash device.

——— **Note** ———

Programming your flash device can take a long time!

The **Program Devices** dialog and the main **Console** view, update as each programming operation completes. On completion, the results are displayed in the **Flash Programmer Results** dialog as shown in Figure 7-3.

———————————



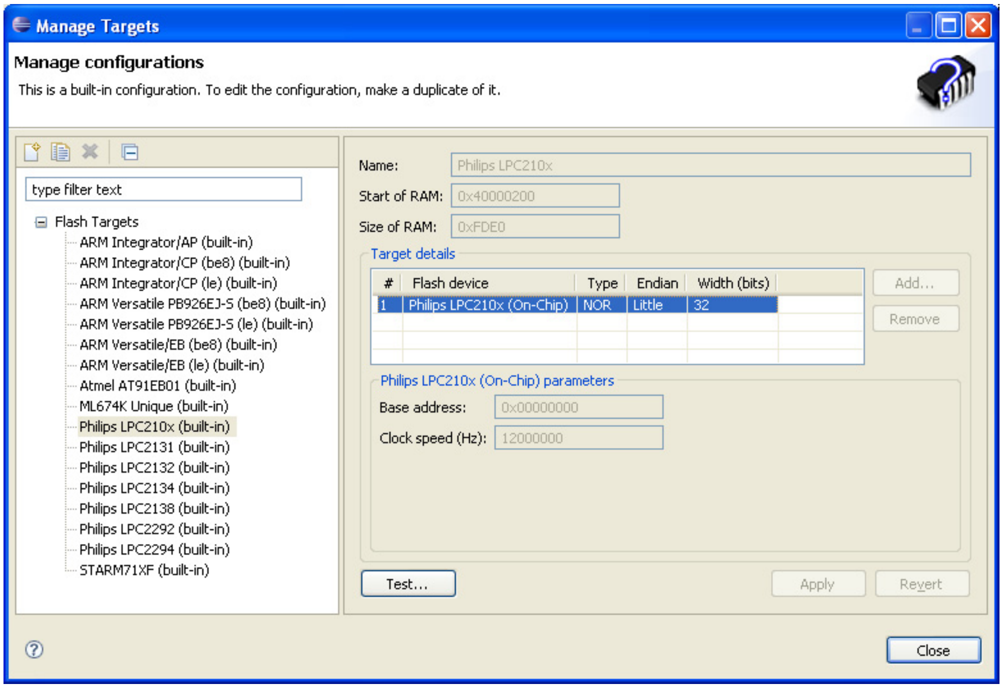
Programming is complete:
25KB image indexed in (HH:MM:SS) 00:00:03
Connected to board in (HH:MM:SS) 00:00:00
Device programmed in (HH:MM:SS) 00:00:21
1 block totalling 128KB programmed in: (HH:MM:SS) 00:00:27

Figure 7-3 Results from programming your device

7.3 Managing flash targets

To access a flash device from the Eclipse IDE you must setup 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 **Target** → **Manage Targets...** from the Eclipse main menu. Figure 7-4 shows the **Manage Targets** dialog.



Manage Targets

Manage configurations

This is a built-in configuration. To edit the configuration, make a duplicate of it.

type filter text

Flash Targets
- ARM Integrator/AP (built-in)
- ARM Integrator/CP (be8) (built-in)
- ARM Integrator/CP (le) (built-in)
- ARM Versatile PB926EJ-S (be8) (built-in)
- ARM Versatile PB926EJ-S (le) (built-in)
- ARM Versatile/EB (be8) (built-in)
- ARM Versatile/EB (le) (built-in)
- Atmel AT91EB01 (built-in)
- ML674K Unique (built-in)
- Philips LPC210x (built-in)
- Philips LPC2131 (built-in)
- Philips LPC2132 (built-in)
- Philips LPC2134 (built-in)
- Philips LPC2138 (built-in)
- Philips LPC2292 (built-in)
- Philips LPC2294 (built-in)
- STARM71XF (built-in)

Name: Philips LPC210x
Start of RAM: 0x40000200
Size of RAM: 0xFDE0

Target details

#	Flash device	Type	Endian	Width (bits)
1	Philips LPC210x (On-Chip)	NOR	Little	32

Philips LPC210x (On-Chip) parameters
Base address: 0x00000000
Clock speed (Hz): 12000000

Figure 7-4 Manage configurations

1. Select your flash target configuration from the tree. If you need to create a new configuration, click on the toolbar or right-click on an existing configuration, and select **New** or **Duplicate**.

2. Setup the target details table and flash device parameters as required to complete this process. For information on each configuration option, access the F1 online help. See *Accessing online help* on page 1-6 for more information.

   Click **Apply** to save the current configuration. This does not connect to your flash device.

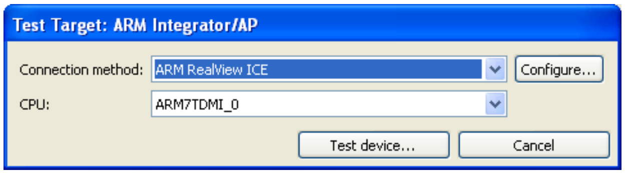
   Click **Revert** to undo any changes and revert to the last saved configuration.

3. If you need to test your device, click **Test...** to access the **Test Target** dialog as shown in Figure 7-5.
   a. Click **Configure...** to setup the **Connection method** and **CPU** settings if you have not already set them up.
   b. Click **Test device...** to erase, program, and validate the required blocks.

   Note

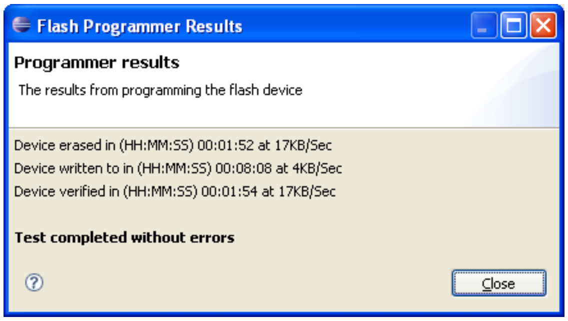
   Testing your flash device can take a long time!

   The **Progress Information** dialog and the main **Console** view update as each programming operation completes. On completion, the results are displayed in the **Flash Programmer Results** dialog as shown in Figure 7-6.



Connection method: ARM RealView ICE
CPU: ARM7TDMI_0

Figure 7-5 Test target connection method



Programmer results

The results from programming the flash device

```
Device erased in (HH:MM:SS) 00:01:52 at 17KB/Sec
Device written to in (HH:MM:SS) 00:08:08 at 4KB/Sec
Device verified in (HH:MM:SS) 00:01:54 at 17KB/Sec

Test completed without errors
```

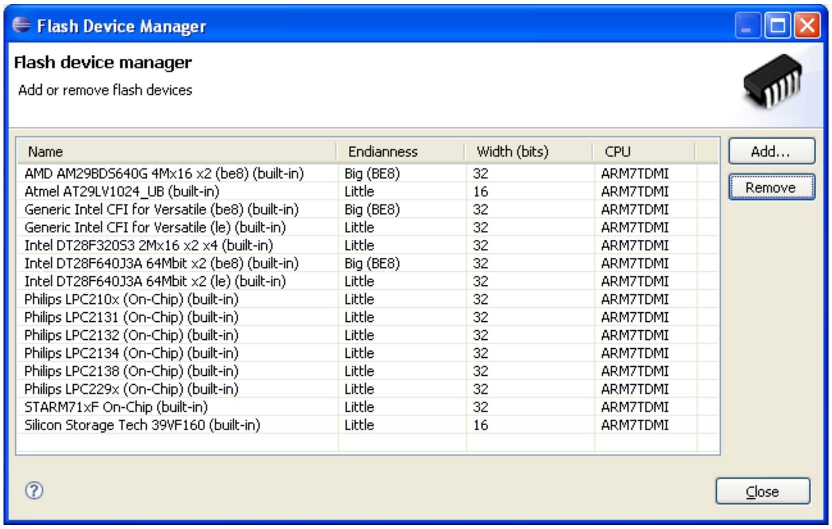
Figure 7-6 Results from testing your device

7.4 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 or export it for use with RealView Debugger.

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

1. Select **Target** → **Flash Device Manager** from the Eclipse main menu, see Figure 7-7.



Name	Endianness	Width (bits)	CPU
AMD AM29BDS640G 4Mx16 x2 (be8) (built-in)	Big (BE8)	32	ARM7TDMI
Atmel AT29LV1024_UB (built-in)	Little	16	ARM7TDMI
Generic Intel CFI for Versatile (be8) (built-in)	Big (BE8)	32	ARM7TDMI
Generic Intel CFI for Versatile (le) (built-in)	Little	32	ARM7TDMI
Intel DT28F320S3 2Mx16 x2 x4 (built-in)	Little	32	ARM7TDMI
Intel DT28F640J3A 64Mbit x2 (be8) (built-in)	Big (BE8)	32	ARM7TDMI
Intel DT28F640J3A 64Mbit x2 (le) (built-in)	Little	32	ARM7TDMI
Philips LPC210x (On-Chip) (built-in)	Little	32	ARM7TDMI
Philips LPC2131 (On-Chip) (built-in)	Little	32	ARM7TDMI
Philips LPC2132 (On-Chip) (built-in)	Little	32	ARM7TDMI
Philips LPC2134 (On-Chip) (built-in)	Little	32	ARM7TDMI
Philips LPC2138 (On-Chip) (built-in)	Little	32	ARM7TDMI
Philips LPC229x (On-Chip) (built-in)	Little	32	ARM7TDMI
STARM71xF On-Chip (built-in)	Little	32	ARM7TDMI
Silicon Storage Tech 39VF160 (built-in)	Little	16	ARM7TDMI

Figure 7-7 Add or remove flash devices

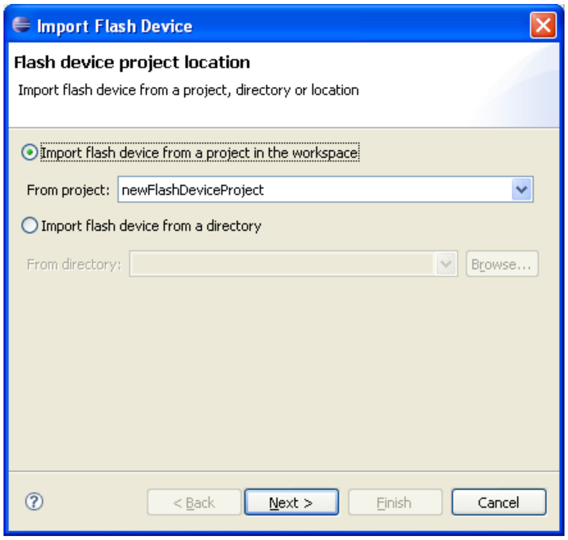
2. Click **Add...** to add a new flash algorithm to your workspace or click **Remove** to remove a flash algorithm from your workspace.

   Note

   You cannot remove a built-in flash algorithm.

3. Select the location of your flash device project, see Figure 7-8 on page 7-9.

   Click **Next**.


Import Flash Device
Flash device project location
Import flash device from a project, directory or location

[✓ Import flash device from a project in the workspace]

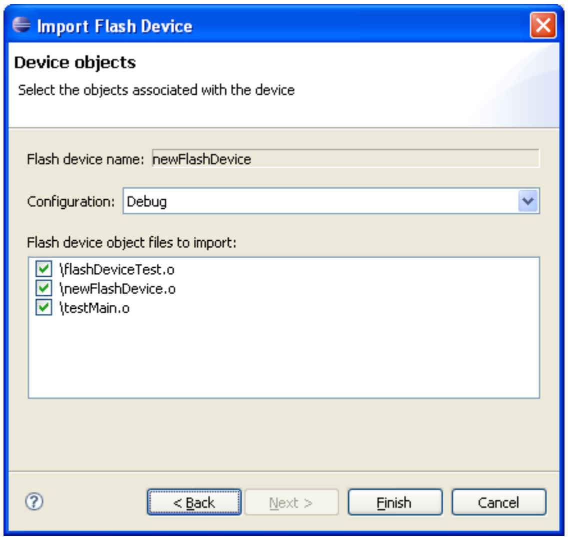
From project: newFlashDeviceProject

Import flash device from a directory

From directory:
Browse...
< Back
Next >
Finish
Cancel

Figure 7-8 Import a flash device

4. Select the configuration and object files associated with your flash device. Click **Finish**.



Flash device name: newFlashDevice

Configuration: Debug

Flash device object files to import:

- ✓ \flashDeviceTest.o
- ✓ \newFlashDevice.o
- ✓ \testMain.o

Figure 7-9 Import flash device object files

7.5 Importing a flash image

To use the ARM Flash Programmer, a flash image must exist in an Eclipse project or folder.

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

1. Select **File** → **Import...** from the Eclipse main menu and then select **Flash Programmer** → **Flash Image**, see Figure 7-10. Click **Next**.

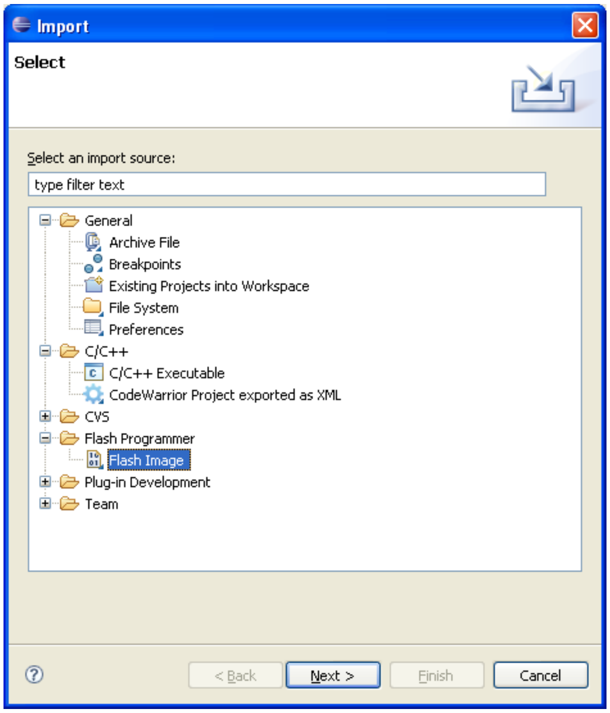
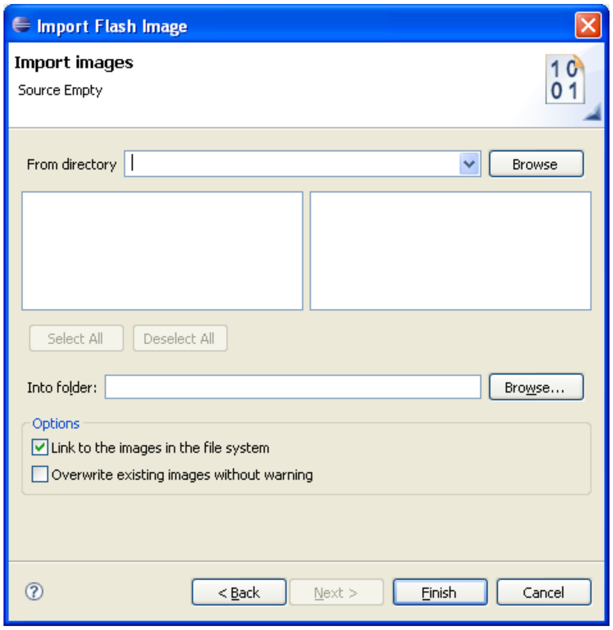


Figure 7-10 Select flash image to import

2. Select the source directory and project folder to store or link the image. Change the **Options** as required and click **Finish**.



✓ Link to the images in the file system

☐ Overwrite existing images without warning

Figure 7-11 Flash image import settings

7.6 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 Eclipse managed make project.

This section describes how to create a new flash algorithm:

1. Select **File** → **New** → **Project...** from the Eclipse main menu.
2. Select **Flash Programmer** → **New Flash Device Project for ARM RVDS**, see Figure 7-12. Click **Next**.

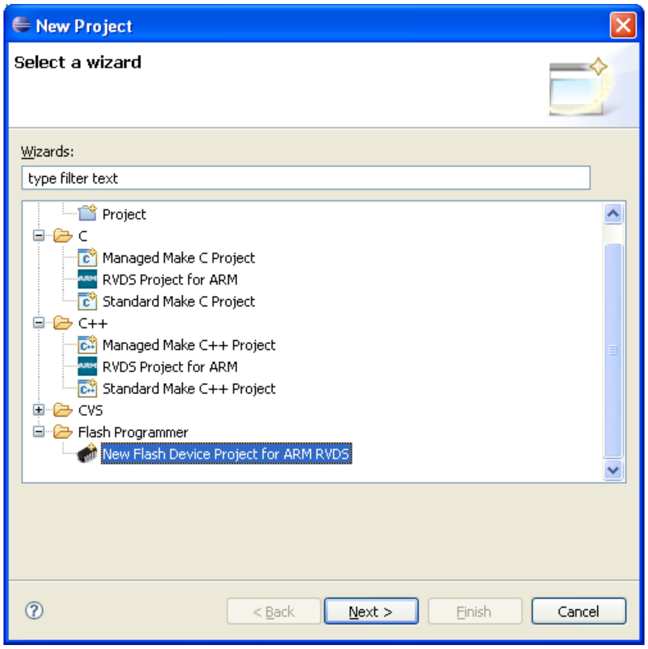
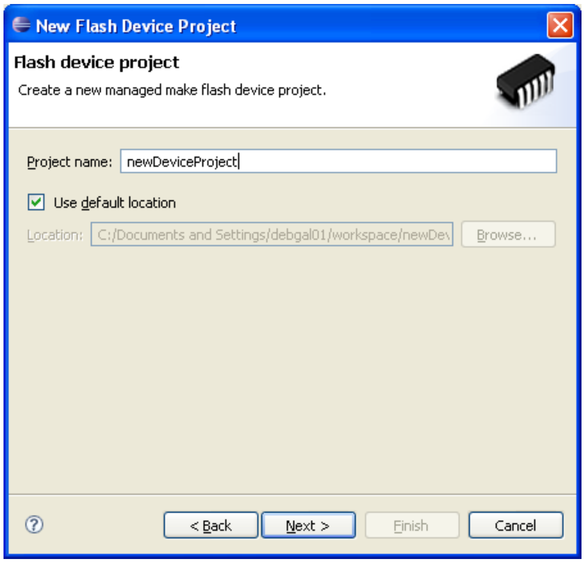


Figure 7-12 New flash device project

3. Enter a name for your flash device project, see Figure 7-13 on page 7-13.



Create a new managed make flash device project.

Project name: newDeviceProject

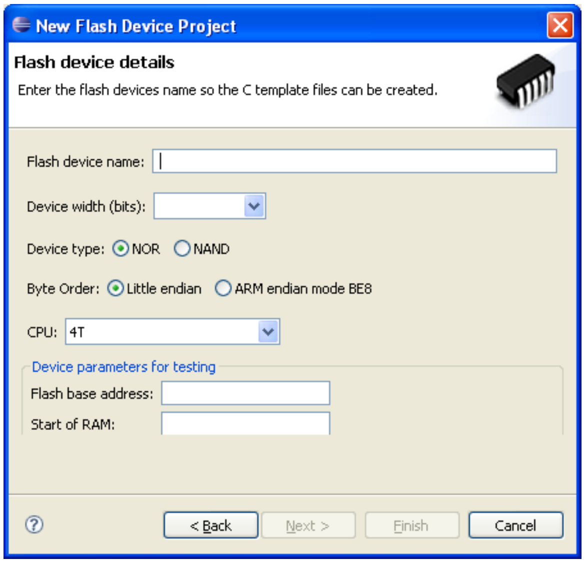
☑ Use default location

Location: C:/Documents and Settings/debgal01/workspace/newDe\

Figure 7-13 Naming your flash project

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

5. Enter the **Flash Device Details** for your project as appropriate, see Figure 7-14 on page 7-14. Click **Next**.



Flash device details

Enter the flash devices name so the C template files can be created.

Flash device name:

Device width (bits):

Device type: [✓ NOR] NAND

Byte Order: [✓ Little endian] ARM endian mode BE8

CPU: 4T

Device parameters for testing

Flash base address:

Start of RAM:

Figure 7-14 Flash device details

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. Click **Finish**.

   Note

   Project settings can be changed using the **C/C++ Build** and **Flash Device** panels in the **Properties** dialog.

7. The new flash device project is visible in the **C/C++ Projects** view.

7.7 Exporting a flash device to 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).

1. Select **File** → **Export...** from the Eclipse main menu and then select **Flash Programmer** → **RealView Debugger Flash Device**, see Figure 7-15. Click **Next**.

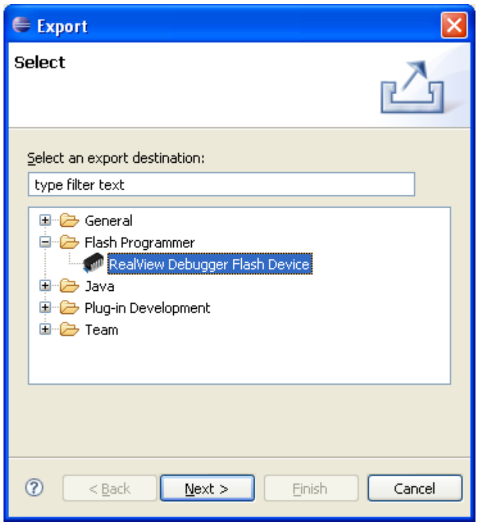
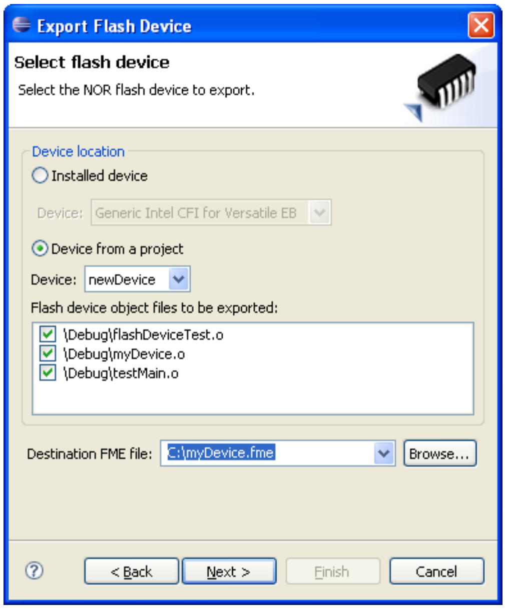


Figure 7-15 Export flash device to RealView Debugger

2. Select the flash device to export and enter the details or browse to the destination FME file, see Figure 7-16 on page 7-16. Click **Next**.



Select the NOR flash device to export.

Device location

( ) Installed device

Device: Generic Intel CFI for Versatile EB

(●) Device from a project

Device: newDevice

Flash device object files to be exported:

- ✓ \Debug\flashDeviceTest.o
- ✓ \Debug\myDevice.o
- ✓ \Debug\testMain.o

Destination FME file: C:\myDevice.fme

Figure 7-16 Export flash device object files

3. Enter the block structure for your flash device using **Add...** and **Edit...** . Alternatively, you can click on **Query device for structure...** to automatically read the block structure from the flash algorithm. Click **Next**.

   Figure 7-17 on page 7-17, Figure 7-18 on page 7-17, Figure 7-19 on page 7-17, and Figure 7-20 on page 7-18 shows the device structure, block and parameter dialogs. Enter the required details and click **Next** to continue to the next dialog.

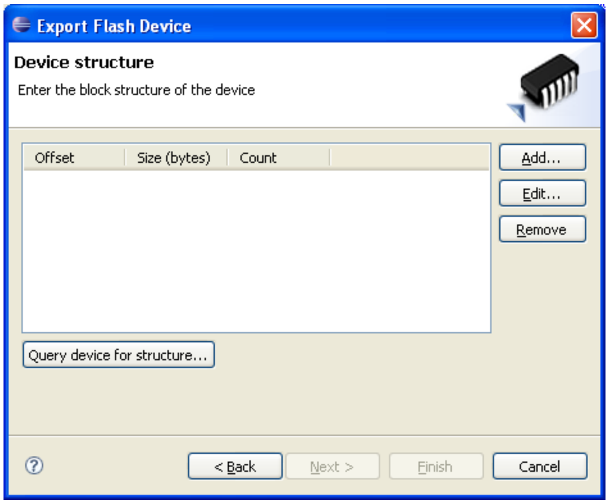
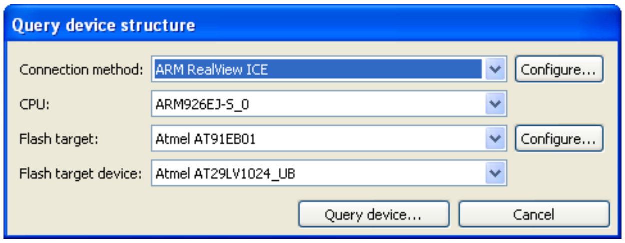


Figure 7-17 Export flash device structure



Connection method: ARM RealView ICE
CPU: ARM926EJ-S_0
Flash target: Atmel AT91EB01
Flash target device: Atmel AT29LV1024_UB

Figure 7-18 Query device structure

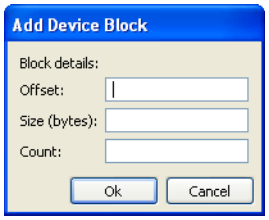
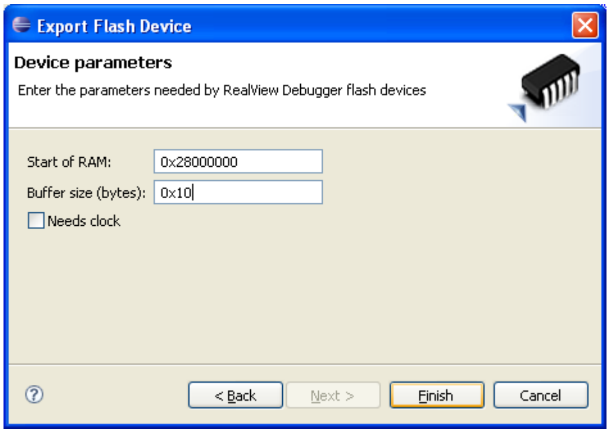


Figure 7-19 Export flash device block



Device parameters

Enter the parameters needed by RealView Debugger flash devices

Start of RAM: 0x28000000

Buffer size (bytes): 0x10

Needs clock

Figure 7-20 Export flash device parameters

4. Click **Finish** to complete the export process and save the FME file.

Note

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

Chapter 8
Working with RealView Debugger

This chapter describes how to launch RealView® Debugger from the Eclipse IDE, and how to configure the connection settings of RealView Debugger. It contains:

8.1 Loading your executable image into RealView Debugger

The Eclipse Plug-ins for RVDS enable target configurations to be setup and saved with your project files. These configuration files are stored in the `rvd` sub-directory within your project.

If your project contains target configuration files, the connection details are automatically transferred to RealView Debugger. See *Load using an existing Eclipse target configuration* for more information.

If your project does not contain target configuration files then you must manually setup a connection to your target. See *Load without an Eclipse target configuration* on page 8-3 for more information.

8.1.1 Load using an existing Eclipse target configuration

Ensure that RealView Debugger is closed before launching from Eclipse to run or debug your executable image. When you launch RealView Debugger from Eclipse 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 Eclipse, ensure your project is built and contains an executable image (see *Building the project* on page 2-10).
2. Right-click on your executable image file. From the context menu, select **Debug As → Load into RealView Debugger**.
3. For a simulated target, Eclipse 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.

Eclipse remembers the last loaded executable image. If you wanted to reload the executable from the same project, you can simply press F11 on the keyboard. The project gets rebuilt if necessary, and the executable image gets 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 Eclipse.

————————

8.1.2 Load without an Eclipse target configuration

You can launch RealView Debugger from Eclipse 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 Eclipse, 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 Eclipse, ensure your project is built and contains an executable image (see *Building the project* on page 2-10).
5. Right-click on your executable image file. From the context menu, select **Debug As → Load into RealView Debugger**.
6. Eclipse launches RealView Debugger automatically and loads your executable image into it.

Eclipse remembers the last loaded executable image. If you wanted to reload the executable from the same project, you can simply press F11 on the keyboard. The project gets rebuilt if necessary, and the executable image gets reloaded into RealView Debugger.

8.2 Creating your debug configuration

You can create and setup your own debug configuration for each executable image in Eclipse. To create a new debug configuration for your executable image:

1. Select your project from the **C/C++ Projects** view.
2. Select **Run** → **Debug** from the Eclipse main menu to show the **Create, manage, and run configurations** dialog.
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** text box.

   Note

   If your project does not contain an executable image, or contains more than one executable image, then the C/C++ Application text box remains blank. Eclipse warns that the program does not exist and disables the **Debug** button on the panel. See *Selecting a different image to debug* on page 8-5, 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.

8.3 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 8-4, to create a new configuration. This section describes how to set up an existing debug configuration.

8.3.1 Selecting an existing debug configuration

To select an existing debug configuration:

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

The Eclipse Plug-ins for RVDS provides four tabbed panels to setup or modify the selected RealView Debugger configuration. The **Main**, **Arguments**, and **Connection** tabs are described in the following sections.

8.3.2 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** text box, or click **Browse...** and select from the list of available projects.

Note

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

C/C++ Application

You can either type the name of the image you want to debug, in the **C/C++ Application** text box, or click **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 will appear in this list.

8.3.3 Configuring RealView Debugger connection settings

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

Connections

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 **Get Connections** for Eclipse 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 checked, Eclipse loads the executable image into the target that is shown first in the list of available connections.

Parts to load

Use this to select which part 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** text box 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.

8.3.4 Specifying execution arguments

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

Program Arguments

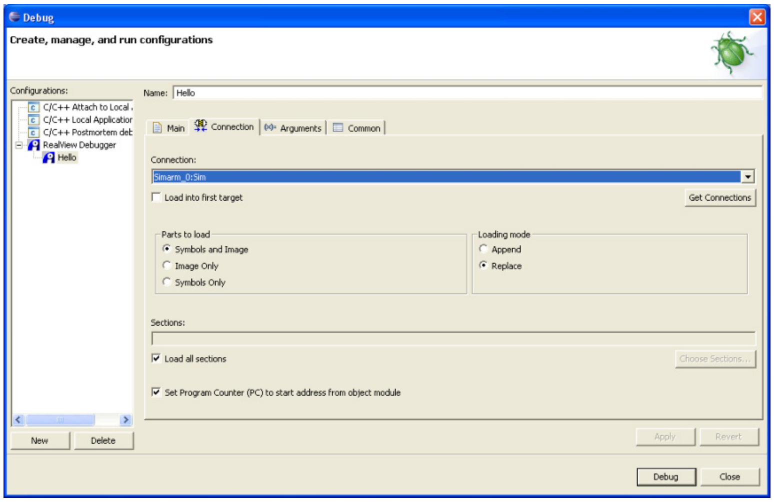
You can specify a space-separated list of arguments to the executable, in the **C/C++ Program Arguments** text box.

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

8.4 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 simple steps in *Loading your executable image into RealView Debugger* on page 8-2. If you wish to change any of the debug configuration settings before loading the executable into RealView Debugger:

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 **Run** → **Debug** from the Eclipse main menu.
3. Select your RealView Debugger configuration from the Configurations box or create a new debug configuration. See *Creating your debug configuration* on page 8-4.
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 8-7.
6. Select the **Connections** tab. Click on **Get Connections**. RealView Debugger launches and Eclipse tries to connect to RealView Debugger.
7. Once Eclipse connects to RealView Debugger, return to the **Connection** tab in the Eclipse **Debug** dialog. 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 8-1 on page 8-9). Modify the load settings as your require. See *Configuring RealView Debugger connection settings* on page 8-6.



Configurations:
- C/C++ Attach to Local ...
- C/C++ Local Applicatior
- C/C++ Postmortem deb
- RealView Debugger
  - Hello

Name: Hello

Connection: Simarm_0:Sim

☐ Load into first target

Parts to load: [✓ Symbols and Image] Image Only, Symbols Only

Loading mode: Append [✓ Replace]

Sections:

☑ Load all sections

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

Figure 8-1 Debug panel

8. Click **Debug**. The project rebuilds if necessary.

9. Your executable image is loaded into the target and you can debug it in RealView Debugger.

Appendix A
Eclipse Terminology and Shortcuts

This appendix describes some of the most common Eclipse terms used in the *RealView Development Suite Eclipse Plug-ins User Guide*. Also listed are some of the main keyboard shortcuts for use with the Eclipse plug-ins for RVDS. It includes the following:

A.1 Keyboard shortcuts

This sections describes some of the most common keyboard shortcuts for use with the Eclipse plug-ins for RVDS:

ALT+Left arrow
Go back in navigation history.

ALT+Right arrow
Go forward in navigation history.

CTRL+L Move to a specified line in the current source code.

CTRL+Q Move to the last edited position in the current source code.

CTRL+SHIFT+F
Activates the code style settings in the **Preferences** dialog and apply them to the current file.

CTRL+SHIFT+L
Opens a small page with a list of all keyboard shortcuts.

CTRL+SHIFT+R
Opens the **Open resource** dialog.

CTRL+SHIFT+T
Opens the **Open Type** dialog.

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

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 current file.

CTRL+Spacebar
Provides auto-completion on selected functions in editors.

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.

A.2 Terminology

This sections describes some of the Eclipse terminology used in the *RealView Development Suite Eclipse Plug-ins User Guide*. The list is in alphabetical order:

Block	A small sub-division of a flash device that can be programmed.
Dialog	A small page containing tabs, panels and editable fields prompting the user to enter information.
Editor	A view that controls the visual aspects of source code for a specific file type. For example the C/C++ editor provides highlighting and content assistance for editing `.c` or `.cpp` files. The ARM assembler editor provides highlighting and content assistance for editing for `.s` files.
Panel	A small box in a dialog or tab to group editable fields.
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.
Perspective	A page within workbench containing a set of related views and editors.
Program	A term used to describe the storing of data on a flash device.
Project	A group of related files and folders in a workspace.
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 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 dialogs to guide the user through a common tasks, for example, creating new files and projects.
Workbench	A window containing perspectives, menus and toolbars.
Workspace	An area designated on the users files system to store files and folders related to specific projects.