ARM Developer Suite

Version 1.0.1

Getting Started

ARM

ARM DUI0064B

Release Information

The following changes have been made to this book.

Change History

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

Contents
Getting Started

Preface

This preface introduces the *ARM Developer Suite* (ADS) and its user documentation. It contains the following sections:

About this book

This book provides an overview of the ADS tools and documentation.

Intended audience

This book is written for all developers who are producing applications using ADS. It assumes that you are an experienced software developer.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to ADS. The components of ADS and the printed and online documentation are described.

Chapter 2 *Differences*

Read this chapter for details of the differences between ADS and the ARM Software Development Toolkit.

Chapter 3 *Creating an application*

Read this chapter for a brief overview of how to create an application using the command-line tools or the CodeWarrior IDE.

Chapter 4 *FLEXlm License Management*

Read this chapter for an explanation of the FLEXlm licence management tool for ADS.

Typographical conventions

The following typographical conventions are used in this book:

`typewriter` Denotes text that may be entered at the keyboard, such as commands, file and program names, and source code.

`typewriter italic`
Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM processor signal names.

`typewriter bold`
Denotes language keywords when used outside example code.

Further reading

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

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

See also the ARM Frequently Asked Questions list at `http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

Refer to the following books in the ADS document suite for information on other components of ADS:

- *CodeWarrior IDE Guide* (ARM DUI 0065A)
- *ADS Tools Guide* (ARM DUI 0067A)
- *ADS Debuggers Guide* (ARM DUI 0066A)
- *ADS Debug Target Guide* (ARM DUI 0058A)
- *ADS Developer Guide* (ARM DUI 0056A).

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

- *ARM Architecture Reference Manual* (ARM DUI 0100). This is supplied in Dynatext format, and in PDF format in *install_directory*\PDF\ARM-DDI0100B_armarm.pdf.
- *ARM Applications Library Programmer's Guide* (ARM DUI 0081). This is supplied in Dynatext format, and in PDF format on the CD.
- *ARM ELF specification* (SWS ESPC 0003). This is supplied in PDF format in *install_directory*\PDF\specs\ARM ELFA08.pdf.
- *TIS DWARF 2 specification*. This is supplied in PDF format in *install_directory*\PDF\specs\TIS-DWARF2.pdf.
- *Angel Debug Protocol*. This is supplied in PDF format in *install_directory*\PDF\specs\ADP ARM-DUI0052C.pdf
- *Angel Debug Protocol Messages*. This is supplied in PDF format in *install_directory*\PDF\specs\ADP ARM-DUI0053D.pdf

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

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

Feedback

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

Feedback on the ARM Developer Suite

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

- details of the release you are using
- details of the platform you are running on, such as the hardware platform, operating system type and version
- a small 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 number of the tool, including the version number and build number.

Feedback on this book

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

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

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the *ARM Developer Suite version 1.0.1* (ADS 1.0.1) and describes its software components and documentation. It contains the following sections:

1.1 About the ARM Developer Suite

ADS consists of a suite of applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of RISC processors.

You can use ADS to develop, build, and debug C, C++, or ARM assembly language programs.

1.1.1 Components of ADS

ADS consists of the following major components:

- *Command-line development tools*
- *GUI development tools* on page 1-3
- *Utilities* on page 1-4
- *Supporting software* on page 1-4.

Command-line development tools

The following command-line development tools are provided:

armcc The ARM C compiler. The compiler is tested against the Plum Hall C Validation Suite for ANSI conformance. It compiles ANSI source into 32-bit ARM code.

armcpp This is the ARM C++ compiler. It compiles ISO C++ or EC++ source into 32-bit ARM code.

tcc The Thumb C compiler. The compiler is tested against the Plum Hall C Validation Suite for ANSI conformance. It compiles ANSI source into 16-bit Thumb code.

tcpp This is the Thumb C++ compiler. It compiles ISO C++ or EC++ source into 16-bit Thumb code.

armasm The ARM and Thumb assembler. This assembles both ARM assembly language and Thumb assembly language source.

armlink The ARM linker. This combines the contents of one or more object files with selected parts of one or more object libraries to produce an executable program. The ARM linker creates ELF executable images.

armsd The ARM and Thumb symbolic debugger. This enables source level debugging of programs. You can single-step through C or assembly language source, set breakpoints and watchpoints, and examine program variables or memory.

Rogue Wave C++ library

The Rogue Wave library provides an implementation of the standard C++ library as defined in the *ISO/IEC 14822:1998 International Standard for C++*. For more information on Rogue Wave, see the online HTML documentation on the CD ROM.

support library

The ARM C libraries provide additional components to enable support for C++ and to compile code for different architectures and processors.

GUI development tools

The following *Graphics User Interface* (GUI) development tools are provided:

AXD The ARM Debugger for Windows and UNIX. This provides a full Windows and UNIX environment for debugging your C, C++, and assembly language source.

ADW The ARM Debugger for Windows. This provides a full Windows environment for debugging your C, C++, and assembly language source.

ADU The ARM Debugger for UNIX. This provides a full GUI environment for debugging your C, C++, and assembly language source.

CodeWarrior IDE

The project management tool for Windows. This automates the routine operations of managing source files and building your software development projects.

Utilities

The following utility tools are provided to support the main development tools:

fromELF The ARM image conversion utility. This accepts ELF format input files and converts them to a variety of output formats, including AIF, plain binary, *Extended Intellec Hex* (IHF) format, Motorola 32-bit S-record format, and Intel Hex 32 format.

armprof The ARM profiler displays an execution profile of a program from a profile data file generated by an ARM debugger.

armar The ARM librarian enables sets of ELF format object files to be collected together and maintained in libraries. You can pass such a library to the linker in place of several ELF files.

Flash downloader
Utility for downloading binary images to Flash memory on a development board.

Supporting software

The following support software is provided to enable you to debug your programs, either under simulation, or on ARM-based hardware:

ARMulator The ARM core simulator. This provides instruction-accurate simulation of ARM processors, and enables ARM and Thumb executable programs to be run on non-native hardware. The ARMulator is integrated with the ARM debuggers.

Angel The ARM debug monitor. Angel runs on target development hardware and enables you to develop and debug applications running on ARM-based hardware. Angel can debug applications running in either ARM state or Thumb state.

Supported standards

The industry standards supported by ADS include:

ar UNIX-style archive files are supported by armar.

DWARF2 DWARF2 debug tables are supported by the compilers and linker. The deprecated format DWARF1 has reduced support.

ANSI C The ARM and Thumb compilers accept ANSI C as input. The option `-strict` can be used to enforce strict acceptance.

C++ The ARM and Thumb C++ compilers support a subset of the ISO C++ language.

EC++ The ARM and Thumb C++ compilers support the *Embedded C++* (EC++) informal standard that is a subset of C++.

ELF The ARM tools produce ELF format files. The FromELF utility can translate ELF files into other formats.

CodeWarrior IDE project files

The project manager for ADS is the Metrowerks CodeWarrior Integrated Development Environment.

RDI All debug agents and targets within ADS have been upgraded to RDI 1.51, a new version of the Remote Debug Interface. The debuggers support all the debug agents (for example ARMulator and Remote_A) that are released as part of ADS. They also support Multi-ICE 1.4.

1.2 Supported platforms

This release of the ADS is supported on the following platforms:

- Sun workstations running Solaris 2.5.1 or 2.6
- Hewlett Packard workstations running HP-UX 10.20
- IBM-compatible PCs running Windows 95, Windows 98, or Windows NT 4.

The CodeWarrior IDE is supported on IBM-compatible PCs running Windows 95, Windows 98, and Windows NT 4.

1.3 Printed documentation

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

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

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

1.3.1 ARM publications

This book contains general information about ADS. Other publications included in the suite are:

- *ADS Tools Guide* (ARM DUI 0067A). This book provides reference information for ADS. It describes the command-line options to the assembler, linker, compilers, and other ARM tools in ADS. The book also gives reference material on the ARM implementation of the C and C++ compilers and the C libraries.
- *ADS Developer Guide* (ARM DUI 0056A). This book provides tutorial information on writing code targeted at the ARM family of processors
- *ADS Debuggers Guide* (ARM DUI 0066A). This book has three main parts in which all the currently supported ARM debuggers are described:
  - — Part A describes the graphical user interface components of *ARM eXtended Debugger* (AXD), the most recent ARM debugger and part of the ARM Developer Suite of software. Tutorial information is included to demonstrate the main features of AXD.
  - — Part B describes the *ARM Debugger for Windows* (ADW) and the *ARM Debugger for UNIX* (ADU). These earlier ARM debuggers continue to be fully supported.
  - — Part C describes the *ARM Symbolic Debugger* (armsd).
- *ADS Debug Target Guide* (ARM DUI 0058A). This book provides reference and tutorial information on the debug targets ARMulator and Angel that can be used with the ARM debuggers.
- *CodeWarrior IDE Guide* (ARM DUI 0065A). This book provides tutorial and reference information on the CodeWarrior Integrated Development Environment. The CodeWarrior IDE is used to manage C, C++, and assembly language projects in ADS. The CodeWarrior IDE and guide are available only on Windows.

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

- *ARM Reference Peripheral Specification* (ARM DDI 0062)
- the ARM datasheet or technical reference manual for your hardware device
- *Help and online documentation* on page 1-10 describes documentation that is available in PDF or HTML format.

1.3.2 Further reading

The ADS books are not intended to be an introduction to the ARM assembly language, C, or C++ programming languages. They do not try to teach programming in C or C++, and are not a reference manual for the C or C++ standards. The following books provide general information:

General information

The following book gives general information about the ARM architecture:

- *ARM System Architecture*, Furber, S., (1996). Addison Wesley Longman, Harlow, England. ISBN 0-201-40352-8.

C++ programming

The following books describe the C++ language:

- *ISO/IEC 14882:1998(E), C++ Standard*. Available from the national standards body (for example, AFNOR in France or ANSI in the USA).

The following books provide general C++ programming information:

- Ellis, M.A. and Stroustrup, B., *The Annotated C++ Reference Manual* (1990). Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-51459-1.

  This is a reference guide to C++.

- Stroustrup, B., *The Design and Evolution of C++* (1994). Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-54330-3.

  This book explains how C++ evolved from its first design to the language in use today.

- Meyers, S., *Effective C++* (1992). Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-56364-9.

  This provides short, specific, guidelines for effective C++ development.

- Meyers, S., *More Effective C++* (1996). Addison-Wesley Publishing Company, Reading, Massachusetts. ISBN 0-201-63371-X.

  This is the sequel to *Effective C++*.

C programming

The following books provide general C programming information:

- Kernighan, B.W. and Ritchie, D.M., *The C Programming Language* (2nd edition, 1988). Prentice-Hall, Englewood Cliffs, NJ, USA. ISBN 0-13-110362-8.

  This is the original C specification, updated to cover the essentials of ANSI C.

- Harbison, S.P. and Steele, G.L., *A C Reference Manual* (second edition, 1987). Prentice-Hall, Englewood Cliffs, NJ, USA. ISBN 0-13-109802-0.

  This is a very thorough reference guide to C, including useful information on ANSI C.

- Koenig, A, *C Traps and Pitfalls,* Addison-Wesley (1989), Reading, Mass. ISBN 0-201-17928-8.

  This explains how to avoid the most common traps and pitfalls in C programming. It provides informative reading at all levels of competence in C.

- ISO/IEC 9899:1990, *C Standard*

  This is available from ANSI as X3J11/90-013. The standard is available from the national standards body (for example, AFNOR in France, ANSI in the USA).

1.4 Help and online documentation

Additional information for ADS is available online. The online documentation consists of online help, Dynatext files, PDF files, and HTML files.

The printed documentation for ADS is also available in Dynatext and PDF files. There is also additional documentation available online that is not part of the printed documentation.

1.4.1 Online help

Whenever you run a windowed component of ADS 1.0.1, the **Help** menu is available in the menu bar.

Select **Help→Contents** to see a display of the main help topics available. (CodeWarrior IDE does not have a **Contents** menu. Use the **How to** menu instead.).

You may navigate to a particular page of help in any one of the following ways:

- From the **Contents** tab of the Help Topics screen, do any of the following:
  - — click on a main topic to select it
  - — click on the **Open** button
  - — click on a sub-topic.
- From the **Contents** tab of the Help Topics screen either:
  - — double-click on a main topic book to open it (single-clicking toggles the open or closed status)
  - — click on a sub-topic.
- From the **Index** tab of the Help Topics screen, do any of the following:
  - — type the first few characters of a likely index entry
  - — scroll down the displayed list of index entries until the entry you want is visible
  - — click on the required index entry.
- From the **Find** tab of the Help Topics screen, do any of the following:
  - — type or select key words that may occur anywhere in the help text
  - — select a topic from the displayed list of topics that contain the specified words.
- From any page of help that has a hypertext link to the page you want, click on the highlighted hypertext link.

- Most pages of online help contain help links that can be clicked on:
  - highlighted hot spots with dashed underlining display brief explanations in pop-up boxes
  - highlighted hot spots with solid underlining jump to other related pages of help
  - browse buttons display related pages of help.

Note

Most help selections can be done by key presses or mouse clicks.

Context-sensitive help

Context-sensitive help is frequently available. With the ADS 1.0.1 component running, position the cursor on any field or button for which you need help and press the F1 key on the keyboard. If relevant online help is available it is displayed.

An alternative method of invoking context-sensitive help is to click on the question mark tool in the toolbar, then click on the field or button for which you need help.

1.4.2 Adobe Acrobat Reader

The manuals for ADS 1.0.1 are provided on the CD-ROM in Acrobat Portable Document Format (PDF) files. You must have a copy of Adobe Acrobat Reader installed before you can view them. Acrobat Reader is supplied with ADS 1.0.1, and is also available from the Adobe web site `http://www.adobe.com`.

The following additional PDF documentation is provided with ADS 1.0.1:

- *ARM Architecture Reference Manual* (ARM DUI 0100). This is supplied in Dynatext format and in PDF format in *`install_directory`*`\PDF\ARM-DDI0100B_armarm.pdf`.
- *ARM ELF specification* (SWS ESPC 0003). This is supplied in PDF format in *`install_directory`*`\PDF\specs\ARM ELFA08.pdf`.
- *TIS DWARF 2 specification*. This is supplied in PDF format in *`install_directory`*`\PDF\specs\TIS-DWARF2.pdf`.
- *Angel Debug Protocol*. This is supplied in PDF format in *`install_directory`*`\PDF\specs\ADP ARM-DUI0052C.pdf`
- *Angel Debug Protocol Messages*. This is supplied in PDF format in *`install_directory`*`\PDF\specs\ADP ARM-DUI0053D.pdf`

To consult the manuals:

1. Start Adobe Acrobat Reader.
2. Choose **Open...** from the File menu.
3. Move to the `/pdf` directory located in your installation directory, if you have installed the online documentation. Otherwise, you can view the PDF files directly from the CD.
4. Open the PDF files you wish to view.

For more information on using Adobe Acrobat Reader, choose **Help→Reader Online Help**.

1.4.3 Dynatext

The manuals for ADS 1.0.1 are provided on the CD-ROM in Dynatext files. A viewer for the files is installed.

To display the online documentation, either:

- select **Online Books** from the **ARM Developer Suite v1.0.1** program group
- execute *`install_directory`*`\dtext41\bin\Dtext.exe`

Figure 1-1 shows the Dynatext browser that will be displayed showing a list of available books.

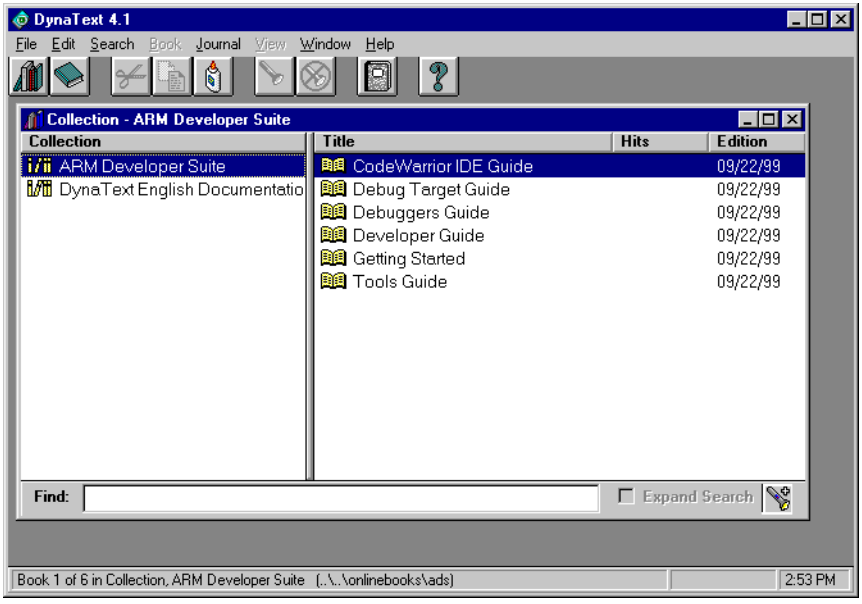

Figure 1-1 Dynatext browser with list of available books

Opening a book

Double-click on a title in the book list to open the book. The table of contents for the book is displayed in the left panel and the text is displayed in the right panel (see Figure 1-2).

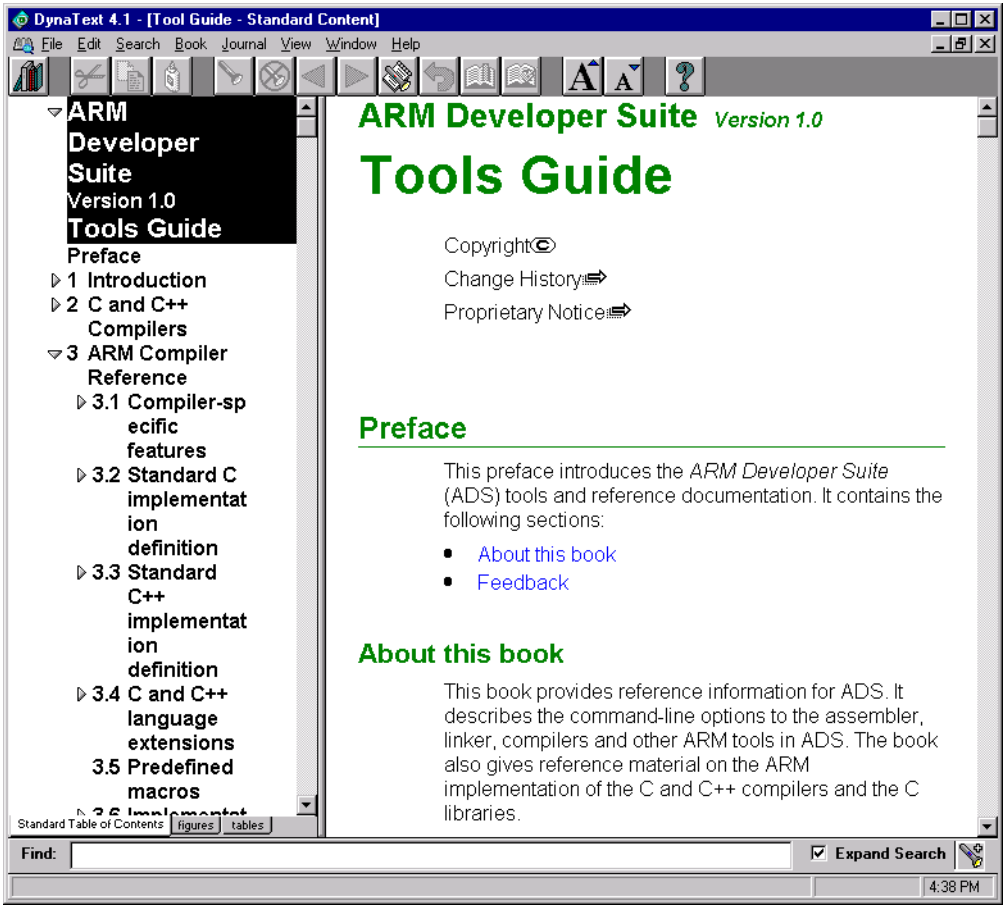

Figure 1-2 Opening a book

Navigating through the book

Click on a section in the table of contents to display the text for that section. For example, selecting *C and C++ libraries* displays the text for that section (see Figure 1-3).

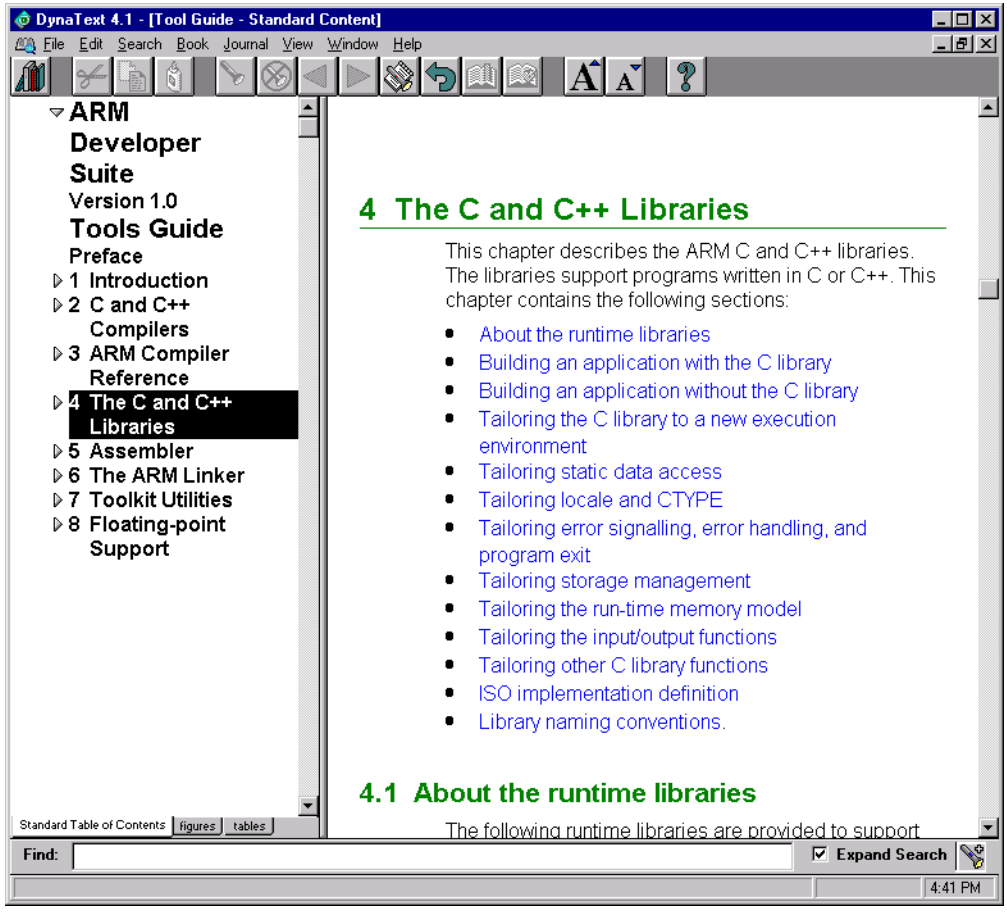

Figure 1-3 Selecting a section from the table of contents

Navigating using hyperlinks

Text in blue indicates a link that will display a different section of a book, or a different book. Plain blue text indicates that the link is within the current chapter. Underlined blue text indicates that the link is either to another chapter within the current book, or to a different book. Hyperlinks behave differently depending on their target:

- if the link is within the current chapter (plain blue text), Dynatext scrolls the current window to display the target
- if the link is to another chapter in the current book, Dynatext opens a new window without a Table of Contents
- if the link is to another book, Dynatext opens a new window with a Table of Contents.

Figure 1-4 shows the browser displaying the text for the linked text.

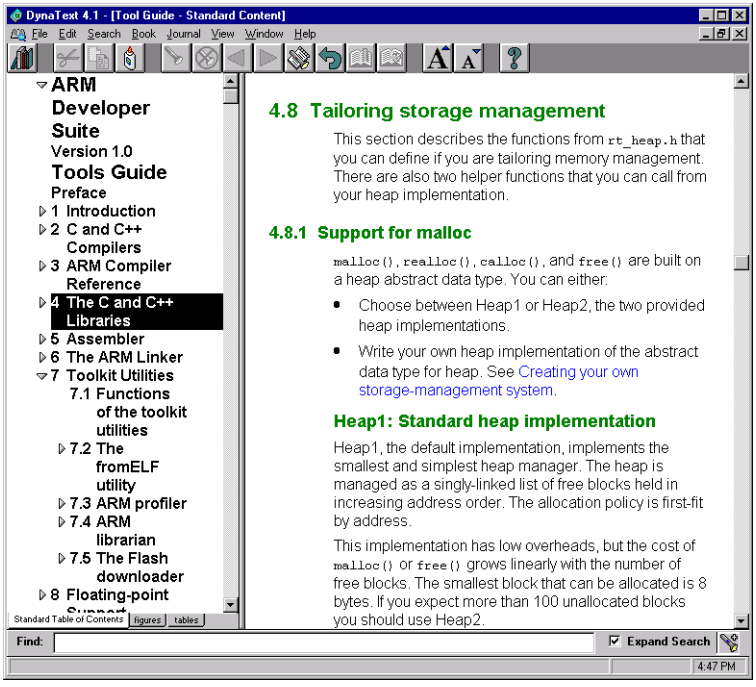

Figure 1-4 Using text links

Displaying graphics

Graphics are not displayed automatically in the Dynatext browser since this might obscure the text. If a graphic symbol is displayed, select it to display the linked graphic in its own window. See Figure 1-5.

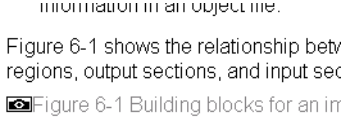

Figure 1-5 Link to a figure

Clicking on the figure icon will display the figure in its own window. See Figure 1-6.

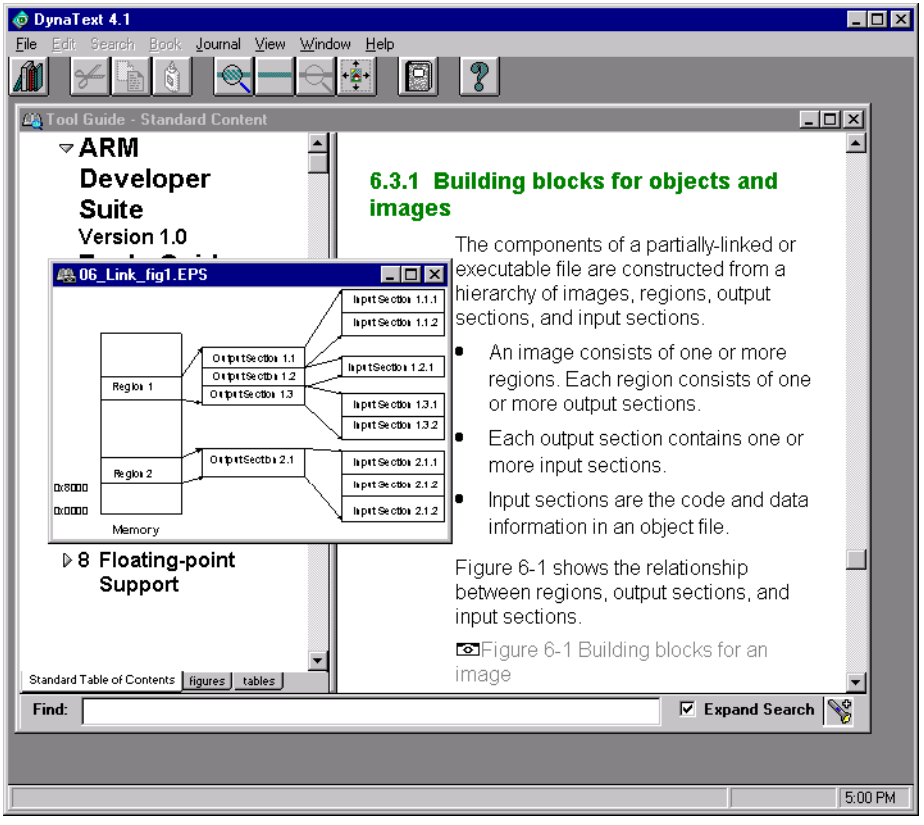

Figure 1-6 Graphic displayed

Navigating to a different book

If the blue link text refers to a different book, clicking on the link text will display the linked book in its own window (see Figure 1-7).

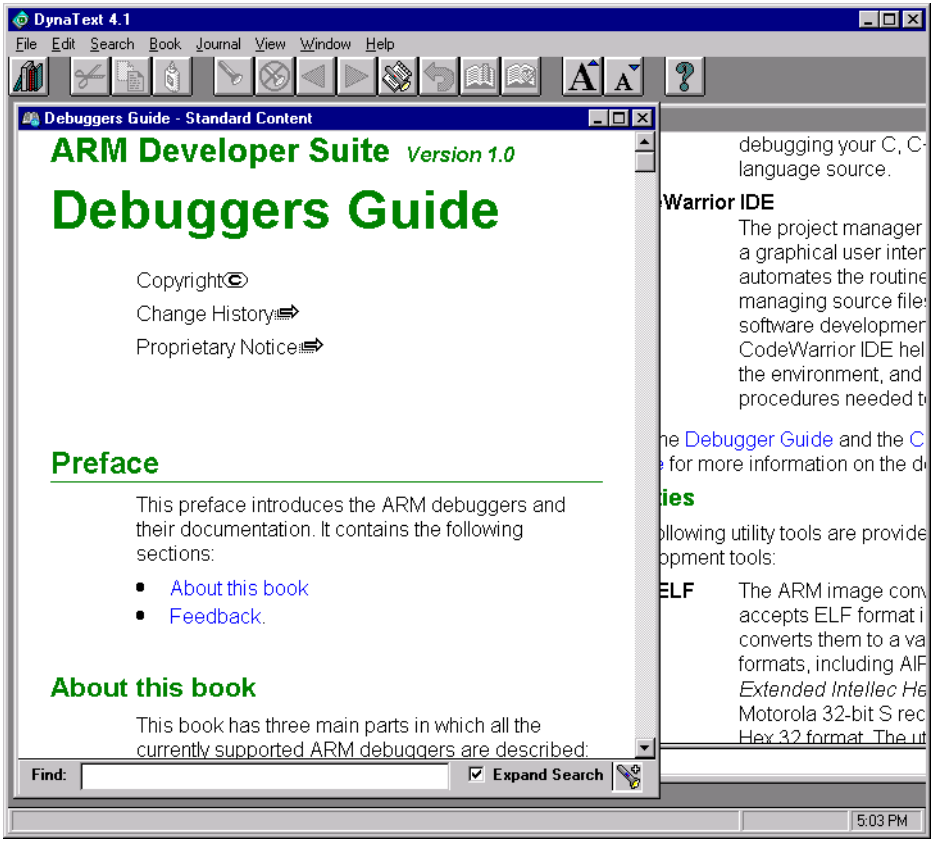

Figure 1-7 Navigating to a different book

Displaying help for Dynatext

Select **Help→Reader Guide** to display help on how to use Dynatext.

1.4.4 HTML

The manuals for the RogueWave C++ library for ADS 1.0.1 are provided on the CD-ROM in HTML files. Use your browser software to view these files. For example, select *install_directory*\Html\stdref\index.htm to display the HTML documentation for RogueWave (see Figure 1-8).

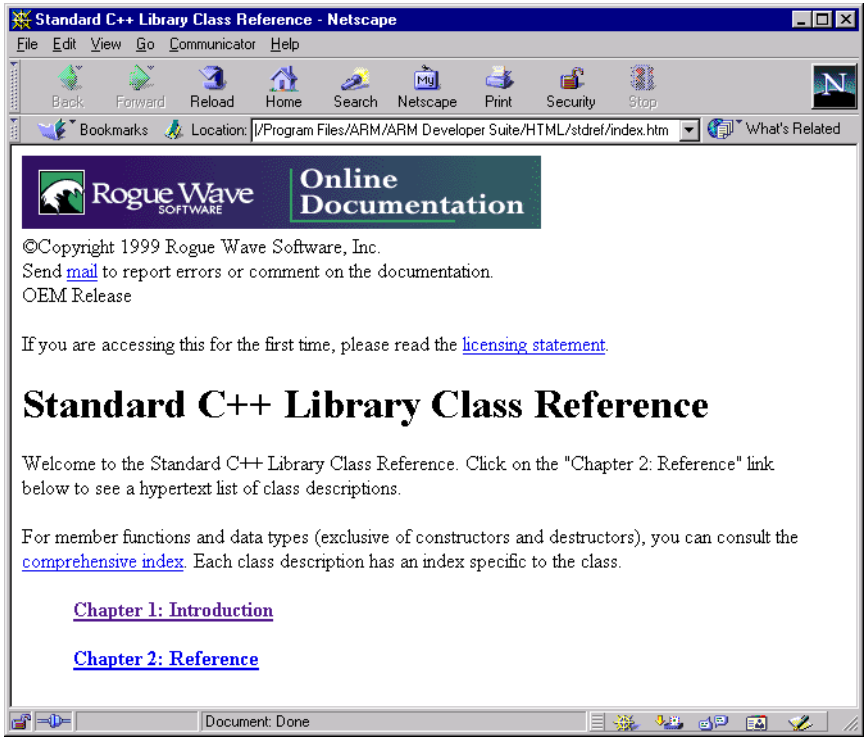

Figure 1-8 HTML browser

Chapter 2
Differences

This chapter describes the major differences between the ARM Developer Suite version 1.0 and the ARM Software Development Toolkit version 2.50:

2.1 Overview

The most important differences between ADS 1.0 and SDT 2.50 are:

- C and C++ libraries are supplied as binaries only. Selection of the appropriate library for the build option is automatic. No rebuild kit or source code is supplied.
- The C libraries are suitable for embedded applications.
- The CodeWarrior IDE is used for project management instead of APM.
- AXD is a new debugger for Windows and UNIX. ADW for Windows and ADU for UNIX are still supported.
- AXD supports the new RDI 1.51 release.
- armar replaces armlib as library manager and ar format replaces ALF as the library format.
- The image format is now ELF.
- The preferred and default debug table format is now DWARF2.
- Support for ARM9E and preliminary support for ARM10.
- Major components are licence managed.
- Manuals are provided in Dynatext form for easy browsing.
- A new ARM/Thumb Procedure Call Standard (ATPCS) encompasses ARM and Thumb on an equal basis.
- The included C++ compilers are fully integrated and improved, and include support for Embedded C++.
- ARMulator supports RPS Interrupt Controller and Timer peripheral models.
- Clearer messages have been provided in many of the tools.

2.2 Functionality enhancements and new functionality

This release of the ADS introduces numerous enhancements and new features. The major changes are described in:

- *Support for new processors (ARM9E and ARM10)*
- *Floating-point support* on page 2-4
- *Debuggers* on page 2-6
- *ARMulator* on page 2-7
- *Angel and Remote_A* on page 2-7
- *Libraries* on page 2-8
- *Library manager* on page 2-9
- *CodeWarrior IDE* on page 2-9
- *Linker* on page 2-10
- *Compilers* on page 2-10
- *Assembler* on page 2-12
- *License management* on page 2-12

2.2.1 Support for new processors (ARM9E and ARM10)

ADS introduces support for the new ARM9E and ARM10 processors.

The new ARM9E instructions are supported by the assembler, the inline assembler of the C and C++ compilers, the debuggers, and the ARMulator.

The new ARM10 instructions are supported by the assembler, the inline assembler of the C and C++ compilers, the debuggers, and the Basic ARM Ten System (BATS) ARMulator model. The compiler performs instruction scheduling for ARM10 code by re-ordering machine instructions to gain maximum speed and minimize wait states. The linker uses `BLX` in interworking veneers when the underlying architecture (the ARM9E and ARM10, for example, have architecture 5) supports it.

2.2.2 New ARM/Thumb procedure call standard

The Procedure Call Standard has been redesigned to:

- give equal emphasis to ARM and Thumb
- interwork between ARM-state and Thumb-state for all variants
- reduce the number of variants
- support position independence
- produce compact code (especially with Thumb)
- be binary compatible with the previous most commonly used PCS variant.

The new ARM/Thumb Procedure Call Standard (ATPCS) enables a consistent ARM and Thumb definition of Read Only Position Independence (also called Position Independent Code), and Read Write Position Independence (also called Position Independent Data) for both ARM and Thumb.

2.2.3 Floating-point support

Enhanced floating-point support is available in the compiler, assembler and debugger:

- The compiler, assembler and debugger support the new VFP floating-point architecture in scalar mode.
- The compiler can generate VFP instructions for double and float operations. (The inline assembler, however, does not support VFP.)
- The assembler supports VFP in vector mode. (New register names and directives are available.)
- The compiler and assembler command-line option `-fpu` specifies the FPA hardware, VFP hardware or software variants.

  Choose `-fpu fpa` or `-fpu softfpa` to retain the old SDT 2.50 format.

Note

The order of the words in a little-endian **`double`** is different for FPA and VFP. If you select `-fpu fpa` or `-fpu softfpa` the SDT 2.50 format is used. If you select `-fpu vfp` or `-fpu softvfp` the new format is used.

There is no functional difference between SoftFPA and SoftVFP. Both implement IEEE floating-point arithmetic by subroutine call, and both use the IEEE encoding of floating-point values into 32-bit words. However, the ordering of the two halves of a **`double`** is different for little-endian code. See *Byte order of long long and double* for details.

2.2.4 Byte order of long long and double

The compilers and assembler now support industry-standard **`long long`** and **`double`** types in both little-endian and big-endian formats. In SDT 2.50, the formats of little-endian **`double`** and big-endian **`long long`** are nonstandard.

If a big-endian 64-bit quantity is represented as `abcdefgh`, with `a` being the most significant byte and `h` the least significant byte, the standard little-endian format is `hgfedcba`. SDT 2.50 used the following non-standard formats:

`efghabcd` For big-endian **`long long`**.

`dcbahgfe` For little-endian **`double`**.

Impact

The format of **long long** is always industry-standard in ADS 1.0. There is no impact if you have used little-endian **long long**. If you previously used big-endian **long long**, you must recompile your code and ensure that it conforms to the new format.

There is no impact if you have used big-endian **double**. If you previously used little-endian **double** and hardware floating-point (FPA), you should continue to use the old little-endian **double** format and select the `-fpu fpa` option in ADS.

If you previously used little-endian double and software floating-point, you can choose whether or not to change to the new format:

- Use `-fpu softfpa` or `-fpu fpa` to retain the old format
- Use `-fpu softvfp` or `-fpu vfp` to use the industry standard format. You must recompile code that defines or references **double** types.

2.2.5 Remote Debug Interface

A new variant of the *Remote Debug Interface* (RDI 1.5.1) is introduced in ADS. The version used in SDT 2.50 was 1.5.

The ADW debugger has been modified to function with RDI 1.0, RDI 1.5 or RDI 1.5.1 client DLLs. AXD works with RDI 1.5.1 targets only.

Debug targets that are released as part of ADS (ARMulators, Remote_A, and Gateway) have been upgraded to RDI 1.5.1.

Impact

Third-party DLLs written to use RDI 1.5 will continue to work with the versions of ADW and armsd shipped with ADS, but will only work with AXD if the DLL is, and reports itself as, RDI 1.5.1 capable.

Third-party debuggers will fail to work with the ADS ARMulators, Remote_A, and Gateway DLLs unless the debuggers conform to RDI 1.5.1.

2.2.6 Debuggers

A new debugger, AXD, is available for use on Windows or UNIX in addition to the existing ADW and ADU. ADW has been enhanced.

All debug agents and targets in ADS support RDI 1.51, a new version of the Remote Debug Interface. The debuggers support all the debug agents (for example ARMulator and Remote_A) that are released as part of ADS. In addition, all debuggers except armsd support Multi-ICE 1.4:

- ADW supports all ADS debug agents, Multi-ICE 1.3, and Multi-ICE 1.4
- ADU supports all ADS debug agents, and Multi-ICE 1.4
- Armsd supports all ADS debug agents
- AXD supports all ADS debug agents and Multi-ICE 1.4.

AXD

The new debugger provides a modern GUI with improved window management, data display, and data manipulation. The debugging views have been redesigned to make the display more relevant to the data. This includes in-place expansion, in-place editing and validation, data sensitive formatting and editing, coloring modified data, and greater user control over formatting and structure.

ADW

ADW enhancements are:

- Support for VFP floating-point opcodes and registers.
- Improved stack-unwinding due to the use of DWARF2 descriptions. In ADS, all standard library functions carry DWARF frame unwinding descriptions with them. These are always generated by ADS compilers and there is new assembler support in ADS to facilitate their generation for hand-written assembly language.

Impact

AXD can debug RDI 1.5.1 targets only. All ARM-supplied debug targets (Multi-ICE, ARMulator, Remote_A, and gateway) support RDI 1.5.1. For non-ARM debug targets that support RDI 1.5 or RDI 1.0, use ADW instead of AXD.

There is no support for conversion of ADW *obey* files to AXD scripts. If existing obey files are important, use ADW instead.

2.2.7 ARMulator

The ARMulator has been enhanced to support RPS Interrupt Controller and Timer peripheral models (as defined in ARM DDI 0062D). The ARMulator supports the following new processor models:

- ARM9E
- ARM10T
- ARM1020T.

The ARM10 models do not support VFP.

There is also a new stack usage monitor memory model available for all processor models except ARM10T and ARM1020T.

The ARMulator supports RDI 1.5.1.

2.2.8 Angel and Remote_A

Angel and Remote_A enhancements are:

- Remote_A connection supports RDI 1.5.1.
- Improved reliability when semihosting.
- Additional Angel ports and improved integration with uHAL.
- Improved coprocessor support, for example FPA (ARM7500) and VFP (ARM10) coprocessors.
- Support for dynamically loaded hardware drivers for the host on Windows and UNIX.

  Hardware other than serial, parallel, or ethernet ports being can be used to communicate with Angel. The GUI interface for Remote_A is extended into the loaded driver.

2.2.9 Libraries

All Libraries (C, C++, math, and floating-point) are released as a set of object code variants that cover all possible choices of Procedure Call Standard and all processor architecture versions. A limited set of variants is required because the libraries have been restructured to remove the necessity for some combinations. The compilation and linking system has been re-engineered so that the correct library variants are automatically linked in for the chosen compilation options. The linker is able to identify the correct library variant from attributes embedded in the ELF. This re-engineering makes the library variants much easier to use and removes the requirement to rebuild different variants.

The C library has been improved and restructured so that there is no requirement for a separate embedded C library. The C library chapter in the *ADS Tools Guide* describes in detail how to construct target-specific libraries.

New real-time (near constant time) versions of the heap management functions `malloc()`, `free()`, `realloc()`, and `calloc()` are provided.

The floating-point libraries have improved performance and functionality. Two versions are provided:

- The version identified by the files beginning with `f_` conforms to IEEE 754 accuracy standards and meets the floating-point arithmetic requirements of the C and Java language standards.
- The version identified by the files beginning with `g_` provides selectable IEEE rounding modes and full control of IEEE exceptions, but at some performance cost.

The Math library has better accuracy and a wider variety of functions (for example, gamma function, cube root, inverse hyperbolic functions).

2.2.10 Library manager

The library manager is armar. The ARM librarian enables sets of ELF format object files to be collected together and maintained in libraries. You can pass such a library to the linker in place of several ELF files. armar files are compatible with the UNIX archive format ar.

Impact

The linker supports the deprecated ALF library format. Use armar for new libraries and migrate your existing libraries to armar.

2.2.11 CodeWarrior IDE

ARM has licensed the CodeWarrior IDE from Metrowerks and is making this available within ADS. This replaces APM on Windows platforms. (It is not available on UNIX platforms).

The CodeWarrior IDE provides a simple, versatile, graphical user interface for managing your software development projects. You can use CodeWarrior for the ARM Developer Suite to develop C, C++, and ARM assembly language code targeted at ARM processors. The CodeWarrior IDE enables you to configure the ARM tools to compile, assemble, and link your project code.

CodeWarrior IDE configuration dialogs

The CodeWarrior IDE dialog boxes are used to select the new features available in the compilers, assembler, and the linker.

Each selectable option on the dialog boxes has a tool tip that displays the command-line equivalent for the option.

Impact

Existing APM projects are not usable with CodeWarrior. There is no support for conversion of `.apj` files to CodeWarrior projects. Use the CodeWarrior IDE for new projects. Migrate your existing APM projects to use the CodeWarrior IDE.

Check the assembler, compiler, and linker options for your new or migrated projects as the defaults for ADS 1.0 are different from the defaults for the SDT 2.50.

2.2.12 Linker

The major linker enhancements are:

- Support for ELF object code.
- Support for automatic selection of the correct library variant.
- Improved scatter loading features to support new execution region attributes:
  - Position Independent (PI)
  - Relocatable (RELOC)
  - linked at a fixed address (ABSOLUTE)
  - simple Overlay (OVERLAY).
- Direct support for ROPI and RWPI procedure call standard variants.
- Support for outputting symbol definitions from a link step and reading them in a later link step (support for system code at a fixed address).

Impact

Update your projects or makefiles to link with the appropriate options. In most cases you will not have to change your source code to use the new options.

Check the assembler, compiler, and linker options for your new or migrated projects as the defaults for ADS 1.0 are different from the defaults for armlink in SDT 2.50.

See *Changed linker behavior* on page 2-26 and the *ADS Tools Guide* for more information.

2.2.13 Compilers

Extensive improvements have been made to the compilers.

C compilers

The following improvements have been made to the C Compiler:

- Assembly language output generated with the `-S` option to the ARM and Thumb compilers can now more easily be assembled. The compilers add `ASSERT` directives for command-line options such as ATPCS variants and byte order to ensure that compatible compiler and assembler options are used when reassembling the output.
- The inline assembler supports the new ARM9E and ARM10 instructions.
- Instruction scheduling for ARM10 minimizes wait states.
- the new VFP architecture is supported.

New compiler options are provided for:

- controlling warnings
- selecting optimization
- generating position-independent code and position-independent data.

C++ compilers

The C++ compilers included with ADS inherit all the benefits of the C compiler. The following additional improvements have been introduced since C++ version 1.10:

- RogueWave Library 2.01. This includes the RogueWave iostream implementation. The iostream implementation supplied with C++ version 1.10 has been removed. Replace references to `stream.h` and `iostream.h` with `iostream`.
- Support for the EC++ informal standard
- Updated vtables to support ROPI
- Improved template handling.

In addition, improvements have been made to the C++ compilers syntax and semantic checking in both strict and non-strict modes. If previously successful programs now fail to compile, please check their syntax first, before concluding that there is a compiler fault.

Other general improvements are support for:

- **`mutable`**
- **`explicit`**
- covariant return types for left-most inheritance
- pseudo-destructors
- aggregates with allow complicated initializations
- template classes with static data members
- temporary destruction order for arguments to functions
- **`explicit`** casts to **`private`** bases
- inline functions
- better overload resolution
- declarations in conditional statements.

See *Changed compiler behavior* on page 2-17 and the *ADS Tools Guide* for more information.

2.2.14 Assembler

Enhancements to the assembler include:

- the assembler provides support for the latest ARM processors
- the assembler outputs ELF object code.

There are considerable changes to assembler directives. See *Changed assembler behavior* on page 2-23 and the *ADS Tools Guide* for more information.

2.2.15 License management

ADS components are license managed by FLEXlm. (See Chapter 4 *FLEXlm License Management* for more information).

2.3 Differences in default behavior

The differences in the default behavior of ADS compared to SDT 2.50 are described in:

2.3.1 Object and library compatibility

As a consequence of the new features introduced with ADS, ADS object files and libraries are not guaranteed to be compatible with SDT 2.50 object files and libraries. You can link SDT 2.50 objects and libraries with ADS images, but you must ensure that your objects are built with appropriate procedure call standard options, and that the following restrictions are observed:

- You must choose the SDT 2.50 default Procedure Call Standard options when using SDT 2.50 (`/hardfp` excluded), and the ADS 1.0 default Procedure Call Standard options when using ADS.
- In ADS, you must use `-fpu FPA`, `-fpu softFPA`, or `-fpu none`. You cannot use the default option of `-fpu softVFP`.
- The format of big-endian **`long long`** has changed. This means that there is no compatibility between ADS and SDT big-endian code if you use **`long long`**.
- There is no equivalent in ADS to the SDT `-apcs /nofpregargs` option for functions that return a floating-point value. Functions that are built with the `-apcs /nofpregargs` option, but do not return a floating-point value, are compatible with functions declared using the new `__softfp` keyword.
- An SDT 2.50 object and an ADS object will be incompatible if they map the same datum using a **`struct`** type T, whether through use of T * pointers or **`extern`** T, and T contains only fields of short alignment.

  A field has short alignment if its type is:

  — [**`unsigned`**] **`short`** [**`array`**]
  — [**`unsigned`**] **`char`** [**`array`**]
  — a **`short enum`** type
  — a **`struct`** containing only fields of short alignment.

In addition, if you link with SDT 2.50 objects you cannot take advantage of some ADS debug enhancements. In particular, you cannot unwind the stack through SDT 2.50 code.

2.3.2 Entry point used with debugger

When an image with an entry point is loaded:

- the CPSR register is set to the value corresponding to a warm boot
- the IRQ and FIQ flags are set (disabling all interrupts)
- mode is set to Supervisor
- Condition Code flags are unchanged
- the processor executes in ARM state.

If the image contains no entry point, no change is made to the CPSR.

2.3.3 Entry point set by linker option

The `-entry` option sets the entry point for an image. There can be only one instance of the `-entry` option to the linker.

Impact

Multiple `-entry` settings generate an error.

2.3.4 ADW

ADW now defaults to VFP mode for the display of floating-point and double values, and for floating-point registers.

The SDT 2.50 version of ADW allowed some debug target settings to be configured by the debugger tab on the configuration screen (for example, ARMulator memory maps and byte order). For ADS debug targets, this tab is greyed out and the configuration button must be used instead. This button invokes the configuration box in the RDI Target.

Impact

The RDI Target configuration box has been extended to handle memory maps and byte order. The debugger tab is still available for old debug target DLLs.

2.3.5 ARMulator

FPE is now deselected by default in ARMulator. If you need FPE support for armsd, use the command-line option `-FPE`. If you need FPE support for ADW or AXD, select the **FPE** option in the ARMulator configuration dialog.

Map file selection has changed since SDT 2.50. The local/global/none map file selection dialog has been replaced with a single map file selection.

2.3.6 ELF, AIF, Binary AIF, IHF and Plain Binary Image formats

The default, and only supported, image format is ELF.

Impact

The preferred way to generate an image in a non-ELF image (such as plain binary or AIF) is to use the fromELF tool to translate the ELF image into the required format.

2.3.7 Floating-point exceptions

The ADS tools have been changed to conform to the IEEE specification. The SDT2.50 tools set the default response to floating-point Invalid-Operation, Divide-By-Zero and Overflow to be a trap causing program termination. This is contrary to IEEE 754 section 7, that states that “The default response to an exception shall be to proceed without a trap.”

Impact

To restore exception handling to the SDT 2.50 default, make the call shown in Example 2-1 before using any floating-point operations. The call should preferably be at the beginning of `main()`.

Example 2-1

```
#include <fenv.h>
#define EXCEPTIONS (FE_IEEE_MASK_INVALID | FE_IEEE_MASK_DIVBYZERO | \
       FE_IEEE_MASK_OVERFLOW)
__ieee_status(EXCEPTIONS, EXCEPTIONS);
```

2.3.8 Stack unwinding

The compilers now always generate DWARF2 stack-unwinding descriptions. In SDT 2.50 they were only generated if the `-g` option was specified (for debug information). The assembler generates stack-unwinding descriptions if the new frame directives are used. The debuggers rely on the stack-unwinding descriptions for stack backtrace.

Impact

If you want to unwind stacks when debugging assembler code, ensure that you use the new frame directives. Stack-unwinding descriptions are automatically generated by the ADS compilers and are included in the libraries released with ADS, so you need only change assembly language code and legacy SDT2.50 code not compiled with debug information (`-g` option). You can examine disassembled output from the compilers to see how to use the assembler frame directives correctly.

2.3.9 Source directory variable in armsd and ADW

The `$sourcedir` variable used by armsd and ADW defaults to NULL if no value is specified. In addition, the delimiter used to separate multiple pathnames has been changed from a space to a semicolon.

The variable is used only to specify alternative search paths to the debuggers. You must use the following conventions when specifying search paths:

- Enclose the full pathname in double quotes.
- In ADW and armsd under Windows DOS, escape the backslash directory separator with another backslash character. For example:

  ```
  "c:\\mysource\\src1"
  ```

- separate multiple pathnames with a semicolon, not with a space character. For example:

  ```
  "c:\\mysource\\src1;c:\\mysource\\src2"
  ```

  You can also specify long pathnames containing space characters. For example:

  ```
  "c:\\my source\\src1;c:\\my source\\src2"
  ```


2.4 Changed compiler behavior

This section describes compiler behavior that is new, changed, deprecated, or obsolete. Obsolete features are identified explicitly. Their use is faulted in ADS. Deprecated features will be made obsolete in future releases. Their use is warned about in ADS.

2.4.1 New compiler options

The following new warning options are available in the compilers:

`-We` Turn off warnings about pointer casts

`-Wm` Turn off warnings about multi-character **`char`** constants

`-Wo` Turn off warnings about implicit conversion to signed **`long long`**

`-Wq` Turn off warnings about C++ constructor initilization order

`-Wy` Turn off warnings about deprecated features.

Use `-W+`*`option`* to turn a warning on. For example use `-W+e` to turn on warnings about pointer casts.

The following additional new options are available in the compilers:

`-Ono_inline` Disable inlining. This option replaces `-zpdebug_inlines`.

`-memaccess` Specifies the memory attributes of the target system.

`-nostrict` Enables minor extensions to the C and C++ standards.

The changes to the qualifiers to the `-apcs` option are listed in Table 2-1.

Table 2-1 Procedure call standard qualifiers

ADS form	SDT 2.50 equivalent
`[no]interwork`	`[no]interwork`
`[no]ropi`	Not available
`[no]rwpi`	Not available
`[no]swstackcheck`	`[no]swstackcheck`
Obsolete. Now always `nofp`	`[no]fp`
No direct equivalent, use `-fpu softFPA`	`softfp`
No direct equivalent, use `-fpu FPA`	`hardfp`

Table 2-1 Procedure call standard qualifiers (Continued)

ADS form	SDT 2.50 equivalent
Not Available	`[no]fpregargs`
Obsolete. Now always `narrow`.	`narrow`, `wide`
No direct equivalent, use `-rwpi`	`[non]reentrant`

Impact

Update your projects or makefiles to compile with the appropriate options. In most cases you will not have to change your source code to use the new options.

Check the assembler, compiler, and linker options for your new or migrated projects as the defaults for ADS 1.0 are different from the defaults for SDT 2.50.

2.4.2 Obsolete compiler pragmas

The following pragmas from the ARM Software Development Toolkit are not supported in the compiler:

```
check_memory_accesses
optimise_cross_jump
optimise_cse
optimise_multiple_loads
optimise_scheduling
side_effects
continue_after_hash_error
debug_inlines
force_toplevel
include_only_once
```

Impact

If you are creating new applications, there is no impact. If you are recompiling existing applications, ensure that the desired build options are specified to the compiler. Remove any obsolete pragmas from your source code and replace them, where necessary, with equivalent compiler options.

2.4.3 Obsolete compiler options

The following options from the ARM Software Development Toolkit are not supported in the compiler:

`-zp`*name*	Select pragma from command line.
`-zi`*number*	Replaced by `-Ospace` and `-Otime`.
`-gx`*letter*	Replaced by the `-O[0|1|2]` options.
`-dwarf`	Use `-dwarf2` (or `-dwarf1`).
`-aof`	Output AOF.
`-asd`	Output ASD format debug tables.
`-MD`	Generate APM dependency.
`-cfront`	Select Cfront-style C++.
`-pcc`	Select Berkeley PCC.
`-fussy`	Synonym for `-strict`.
`-pedantic`	Synonym for `-strict`.
`-fw`	Make string literals writable.
`-za`*number*	Use `-memaccess` instead. The default behavior for ADS 1.0 is for `LDR` to access only word-aligned addresses (`-za1`).
`-zt`	Fault tentative declarations. This is now the default unless `-strict` is specified.
`-zz`*number*	Default is `-zzt0`.
`-zzt`*number*	Combines the `-zt` and `-zz` options.
`-zap`	Specify whether pointers to structures are assumed to be aligned on at least the minimum byte alignment boundaries set by `-zas`. The behavior for ADS 1.0 is `-zap0`.
`-zat`	Default is `-zat1`.
`-zr`*number*	Set the number of register values transferred by `LDM` and `STM` instructions. The compilers never generate `LDM` or `STM` instructions that transfer more than nine register values for either ARM code or Thumb code.
`-fz`	This is now the default.

Impact

If you are creating new applications, there is no impact. If you are recompiling existing applications, ensure that the desired build options are specified to the compiler. Remove any obsolete options from your make files and replace them, where necessary, with equivalent options. Check the assembler, compiler, and linker options for your new or migrated projects as the defaults for ADS 1.0 are different from the defaults for the SDT 2.50.

2.4.4 Deprecated compiler options

The following options are deprecated and will not be supported in future versions of the compiler:

`-dwarf1` Use `-dwarf2`.

`-proc, -arch` Select processor or architecture. Use `-cpu` instead.

`-zas`*num* Align structures on at least a *num*-byte boundary (1, 2, 4, or 8). The default is now 1 (align only as strictly as the contents of the structure require).

Impact

You can still output DWARF1 debug tables. However, the functionality of these output files when used with the new debuggers might be reduced. Use DWARF2 format for new projects and update your existing tools to use the DWARF2 format.

2.4.5 Obsolete ARM-specific language extensions

The following language extensions are obsolete:

`__global_freg`
: This language extension is not required.

`___weak (three underscores)`
: This was a synonym for `__weak` (two underscores) in SDT 2.50. Use `__weak`.

`__softfp` This is a storage class specifier you can use in the declaration of a function to indicate that the function has a software floating-point interface (a **`double`** parameter passed in two integer registers, a **`double`** result returned in a0, a1) even though its implementation may use floating-point instructions. Use this to create ARM-state, VFP-using (or FPA-using) functions that you can call directly from Thumb state (in which floating-point instructions are inaccessible).

2.4.6 Obsolete and new predefined macros

The obsolete predefined macros are listed in Table 2-2.

Table 2-2 Obsolete predefined macros

Predefine	Status	Comments
__CLK_TCK	Obsolete	C library use only.
__APCS_32	Obsolete	Relates to obsolete APCS/TPCS. No ATPCS equivalent.
__APCS_FPREGARGS	Obsolete	Relates to obsolete APCS/TPCS. No ATPCS equivalent.
__APCS_NOFP	Obsolete	Relates to obsolete APCS/TPCS. No ATPCS equivalent.
__APCS_REENT	Obsolete	Relates to obsolete APCS/TPCS. No ATPCS equivalent.
__APCS_NOSWST	Obsolete	Relates to obsolete APCS/TPCS. Use new __APCS_SWST.
__CFRONT_LIKE	Obsolete	The option `-cfront` is now obsolete.
__DIALECT_PCC	Obsolete	The option `-pcc` is now obsolete.
__DIALECT_FUSSY	Obsolete	The option `-fussy` is now obsolete.

The new predefined macros are listed in Table 2-3.

Table 2-3 New predefined macros

Predefine	Status	Comments
__CC_ARM	New	Always defined.
__STRICT_ANSI__	New	Set by -strict.
__embedded_cplusplus	New	Set by -embeddedcplusplus.
__APCS_ROPI	New	Set by -apcs /ropi.
__APCS_RWPI	New	Set by -apcs /rwpi.
__APCS_SWST	New	Set by -apcs /swst.
__FEATURE_SIGNED_CHAR	New	Set by -zc.
__OPTIMISE_SPACE	New	Set by -Ospace.
__OPTIMISE_TIME	New	Set by -Otime.
__TARGET_FPU	New	Target Floating Point Unit
__TARGET_FEATURE_DSPMUL	New	Set if ARM9E multiplier available.

2.5 Changed assembler behavior

This section describes assembler behavior that is changed, deprecated, or obsolete. Obsolete features are identified explicitly. Their use is faulted in ADS. Deprecated features will be made obsolete in future releases. Their use is warned about in ADS.

2.5.1 New assembler options

The following enhancements and changes are available in the assembler:

- The assembler provides new ATPCS command-line options similar to those for the compilers.
- The default floating-point option is `-fpu softvfp`.
- A new default software stack checking option of `-swstna` is introduced for code that is compatible with both software stack checking code and non software stack checking code. This option makes explicit the behavior of the assembler. There is no change to the default behavior.
- The assembler always outputs ELF object code. AOF is no longer supported.
- The new `-memaccess` option specifies the memory attributes of the target system.
- The `-list` option now accepts an argument of - to select stdout.
- DWARF2 stack-unwinding descriptions can be, and are recommended to be, produced by the use of new directives.
- The assembler supports the new ARM9E and ARM10 instructions. Use one of ARM9E, ARM10TDMI, ARM1020T, or ARM10200 with the `-cpu` option.
- Support is provided for VFP in both scalar and vector mode.
- New directives `DCQ` and `DCQU` define a 64-bit integer value. `DCQ` is aligned to a 32-bit boundary while `DCQU` is unaligned (byte boundary).
- The `DCFD`, `DCFDU`, `DCFS` and `DCFSU` directives now also accept a hex-constant form of operand that specifies the IEEE bit-pattern of the value.
- There are new synonyms `FIELD` and `SPACE` for `#` and `%` directives.
- Directives are now accepted in all upper case, or all lower case, but not a mixture. Previously, only the upper case form was accepted.
- The `EXPORT` directive may have a new attribute, `WEAK`. This defines the exported symbol as a `WEAK` symbol in ELF.

- The semantics of the EXTERN and IMPORT directives have changed and they are no longer synonyms. An unused IMPORT will generate an undefined global symbol, whereas an unused EXTERN will generate no symbol. (In SDT 2.50 an unused EXTERN or IMPORT symbol was made WEAK).
- The AREA directive has a new attribute, ASSOC= *area_name*) that requires this AREA to be included in a link step whenever the associate area (named by the ASSOC=*area_name*) is included. The assembler implements the requirement by generating an R_ARM_NONE relocation at offset 0 of area *area_name*, relative to the section symbol for the area defined by the AREA directive.
- The new directive REQUIRE*area_name* requires *area_name*. to be included in any link step that includes the requiring section. The assembler implements the requirement by generating an R_ARM_NONE relocation in the current section to the required *area_name*.
- The DCD directive now accepts expressions evaluating the difference between a label in another section and a position in the current section.
- The DCW and DCB directives now accept expressions including an external symbol.
- The new DCDO directive treats label operands as sb-relative.
- The literal-using, pseudo-instruction forms of load and store instructions (for example, LDR rx,=yyy) can now take external symbols as immediate values (yyy).
- The ARM instructions of the form *data-processing-op* rd,rn,#sym can now take external symbols as immediate operands.
- ARM and Thumb SWI instructions can now take external symbols as immediate operands.
- If you select a cpu or architecture that does not support Thumb, an attempt to generate Thumb code will generate an error message. For example armasm -cpu 4 will not accept Thumb instructions but armasm -cpu 4T will.

2.5.2 Features of the SDT assembler not supported

The following assembly language features are no longer supported and are faulted:

- AREA directive with attribute ABS, BASED, A32bit, HALFWORD, INTERWORK, PIC, REENTRANT

  - — ABS has been withdrawn because it conflicts with the linker scatter loading mechanism. An AREA previously declared ABS should now be placed using a scatter-loading description
  - — BASED has been withdrawn because it was needed only for the old shared library mechanism that is now obsolete. No workaround is necessary.
  - — A32bit has been withdrawn as it was needed only to distinguish 32 bit mode code from 26 bit mode code and 26 bit mode is now obsolete.
  - — INTERWORK and PIC have been withdrawn as the ATPCS and architecture are now always specified on the command line. Any occurrences of these attributes should be deleted, and replaced by the corresponding new -apcs command line qualifiers.

- value 32 as operand to the ALIGN area attribute. The assembler accepted 32 as operand to ALIGN even though it was not useful. The only address that satisfies ALIGN=32 is 0, and if that is the desired behavior it can be expressed by using a scatter-loading description to place the AREA at address 0
- IMPORT directive with attribute FPREGARGS. The FPREGARGS attribute had no effect and has been removed.
- EXPORT directive with attribute FPREGARGS, and LEAF.
  - — The FPREGARGS attribute had no effect. The workaround is to delete it from assembly source.
  - — The LEAF attribute was needed only for the old shared library mechanism that is now obsolete. The workaround is to remove it.
- ADR pseudo-instructions with out-of-area symbol operands. The workaround is to load out-of-area addresses into registers using LDR.

2.5.3 Deprecated assembler options

The following options are deprecated and will not be supported in future versions of the assembler:

-dwarf1 DWARF1 debug tables will not be supported in future versions.

-proc Select processor (use -cpu instead).

-arch Select architecture (use -cpu instead).

Impact

Use DWARF2 format for new projects and update your existing tools to use the DWARF2 format.

2.6 Changed linker behavior

This section describes linker behavior that is changed, deprecated, or obsolete. Obsolete features are identified explicitly. Their use is faulted in ADS. Deprecated features will be made obsolete in future releases. Their use is warned about in ADS.

2.6.1 New or changed linker behavior

The following new or significantly changed options are available in the linker:

- The linker is now an ELF-only linker.
- The syntax of the `-remove` command has been expanded to include section attribute qualifiers. This is backwardly compatible with SDT 2.50.

  The linker now has `-remove` as its default option. The SDT 2.50 default was `-noremove`. The `-remove` option is strongly recommended with C++ in order to reduce code size. Use the new linker option `-keep` if you want to keep sections that are not referenced.
- The syntax of `-first` and `-last` has been changed to identify both object and section name, not just section name as in SDT 2.50. There is no backward compatibility with SDT 2.50.
- The syntax of the `-entry` command has been changed to allow more flexible selection. Only one entry point can be specified to the linker. There is some backward compatibility with SDT 2.50.
- The `veneers` argument has been added to the `-info` option.

The following new armlink options have been added:

`-partial`	Generate a partially-linked ELF object
`-ropi`	RO execution region is position-independent
`-rwpi`	RW execution region is position-independent
`-split`	Image has two load regions
`-keep`	Specify sections to be retained even if unused
`-locals`	Add local symbols to image symbol table
`-nolocals`	Remove local symbols from image symbol table
`-xreffrom`	List section cross references in image from a section
`-xrefto`	List section cross references in image to a section
`-strict`	Strict compliance to build attribute rules
`-symdefs`	Create, or read, a list of symbol definitions.

2.6.2 Obsolete linker options

The following options from the ARM Software Development Toolkit are not supported in the linker:

`-aof`	Create output in AOF format
`-aif`	Create output in AIF format
`-aif -bin`	Create output in AIF BIN format
`-bin`	Create output in BIN format
`-base`	Alias for ro-base
`-data`	Alias for rw-base
`-dupok`	Allow multiple definitions
`-[no]case`	Case sensitive/insensitive matching
`-match`	Symbol matching options
`-nozeropad`	Do not include ZI section in binary images
`-info interwork`	Output information on interworking
`-u`	Match all unresolved symbols.

Impact

If you are creating new applications, there is no impact. If you are relinking existing applications and libraries, ensure that the desired build options are specified to the assembler, compiler and linker. Remove any obsolete options from your make files and replace them, where necessary, with equivalent options. Check the assembler, compiler, and linker options for your new or migrated projects as the defaults for ADS 1.0 are different from the defaults for the SDT 2.50.

2.7 Obsolete components and standards

This section describes components of SDT 2.50 that are not available in ADS.

2.7.1 APM

APM is not provided.

Impact

Use the CodeWarrior IDE or a make utility.

2.7.2 Armmake

Armmake is not provided. There is no longer a need to rebuild the C Libraries, therefore the ARM-specific make utility has been removed.

Impact

None. Use nmake, make, or gnumake if you want to use a make utility.

2.7.3 Armlib

The ARM librarian, armlib is not provided. It has been replace by a new utility, armar, that creates ELF `ar` files. armar provides similar functionality to armlib, but supports ELF instead of AOF.

2.7.4 Decaof and Decaxf

Decaof and decaxf are not provided.

Impact

The fromELF utility provides equivalent functionality for ELF formats

2.7.5 DWARF1

The compilation tools produce DWARF2 debug table formats by default. The compiler and assembler can still produce DWARF1 for compatibility with third party tools that require DWARF1, although DWARF1 will only support debugging for C compiler images produced with the `-O0` option and will not support debugging of C++ images.

DWARF1 is deprecated and will be removed in a future release of ADS

Impact

Use DWARF2 format.

2.7.6 26-bit addressing

ADS does not support 26-bit addressing. Removal of 26-bit support has enabled a more efficient ATPCS to be designed.

Impact

Continue to use SDT2.50 if you need 26-bit support.

2.7.7 AOF, AIF, IHF, and Plain Binary image formats

The SDT 2.50 linker gave warnings when asked to generate an AIF image, a binary AIF image, an IHF image or a plain binary image. The ADS linker refuses to generate these images and is now a pure ELF linker. Although the linker is capable of inputting AOF files, you are strongly recommended not to link with old AOF files because of changes to both the Procedure Call Standard and changes to debug tables.

Impact

Use the fromelf tool to translate the ELF image into non-ELF formats such as AIF, Plain binary, Extended Intellec Hex (IHF), Motorola 32 bit S-record, Intel Hex 32.

Future releases of the linker will not allow AOF input files.

2.7.8 RDI 1.50

A new variant of the Remote Debug Interface (RDI 1.5.1) is introduced in ADS. The version used in SDT 2.50 was 1.5. See *Debuggers* on page 2-6 for details of RDI 1.51.

Chapter 3
Creating an Application

This chapter describes how to create an application using ADS. This chapter contains:

3.1 Using the CodeWarrior IDE

The CodeWarrior IDE provides a simple, versatile, graphical user interface for managing your software development projects. You can use CodeWarrior for the ARM Developer Suite to develop C, C++, and ARM assembly language code targeted at ARM processors. The CodeWarrior IDE enables you to configure the ARM tools to compile, assemble, and link your project code.

The CodeWarrior IDE enables you to organize source code files, library files, other files, and configuration settings into a *project*. Each project enables you to create and manage multiple *build targets*. A build target is the collection of build settings and files that determines the output that is created when you build your project. Build targets can share files in the same project, while using their own build settings.

Note

A build target is distinct from a *target system*, such as an ARM development board. For example, you can compile a debugging build target and an optimized build target of code targeted at hardware based on an ARM7TDMI.

CodeWarrior for the ARM Developer Suite provides preconfigured *Project stationery* files for common project types, including:

- ARM Executable Image
- ARM Object Library
- Thumb Executable Image
- Thumb Object Library
- Thumb/ARM Interworking Image.

You can use the project stationery as a template when you create your own projects.

The non-interworking ARM project stationery files define three build targets. The Interworking project stationery defines an additional three build targets to compile Thumb-targeted code. The basic build targets for each of the stationery projects are:

Debug This build target is configured to build output binaries that are fully debuggable, at the expense of optimization.

Release This build target is configured to build output binaries that are fully optimized, at the expense of debug information.

DebugRel This build target is configured to build output binaries that provide adequate optimization, and give an adequate debug view.

For more information on using the CodeWarrior IDE to create complex, dependent build target relationships, see the *CodeWarrior IDE Guide*.

3.1.1 Creating and building a simple project

Note

This example assumes that you installed the example code supplied with ADS 1.0, and that you have installed in the default installation directory. Example code is installed by default unless you have chosen a minimal install or a custom install.

This section describes how to create and build a simple project. It uses source files from the `dhryansi` example supplied with ADS 1.0 to give an introduction to configuring tool options and using build targets in the CodeWarrior IDE.

The following sections give a summary of how to:

- Create a new project using ARM project stationery
- Add source files to your project
- Configure the build target settings for your project
- Compile and link an executable image.
- Execute the AXD debugger to debug your image.

Creating a new project from ARM project stationery

To create a new project, and compile and link an application using the CodeWarrior IDE:

1. Select **Programs → ARM Developer Suite → CodeWarrior for ARM Developer Suite v1.0** from the Windows **Start** menu to start the CodeWarrior IDE.
2. Select **New…** from the **File** menu. A New dialog is displayed (Figure 3-1 on page 3-4).

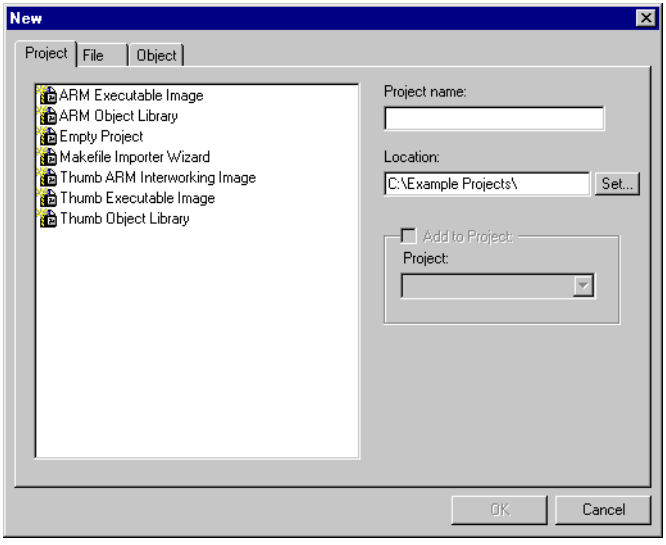

Figure 3-1 New dialog

3. Ensure that the **Project** tab is selected. The available ARM project stationery is listed in the left of the dialog (see Figure 3-1 on page 3-4), together with the Empty Project stationery and the Makefile Importer wizard.

   See the *CodeWarrior IDE Guide* for more information on using empty projects and the Makefile Importer wizard.

4. Select **ARM Executable Image** from the list of project stationery.

5. Click the **Set…** button next to the Location field. A Create New Project dialog is displayed (Figure 3-2 on page 3-5).

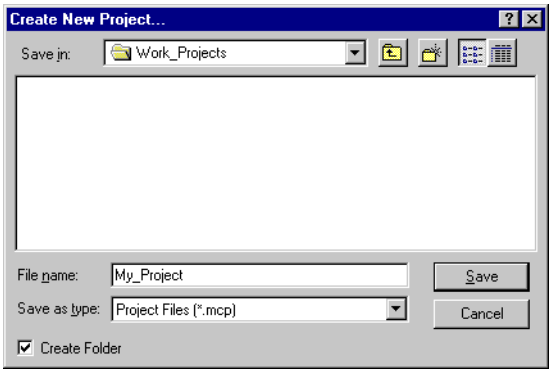

Figure 3-2 Create New Project dialog

6. Navigate to the directory where you want to save the project and enter a project name, for example `My_Project`. Leave the **Create folder** checkbox selected.

7. Click **Save**. The CodeWarrior IDE sets the Project Name field and Location path in the New dialog box. The Location path is used as a default when you create additional projects.

8. Click **OK**. The CodeWarrior IDE creates a new project based on the ARM Executable Image project stationery, and displays a new project window with the Files view selected (Figure 3-3).

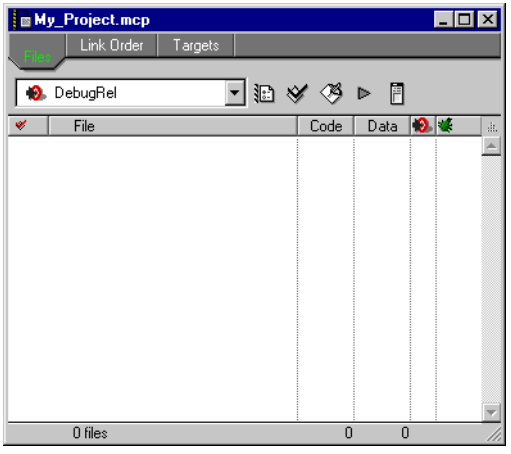

Figure 3-3 New project

Adding source files to the project

Projects created from ARM project stationery do not contain source files. This section describes how to add the source files from the `dhryansi` example.

To add source files to a project:

1. Ensure that the project window is the active window.
2. Select **Add Files…** from the **Project** menu. A Select files to add… dialog is displayed.
3. Navigate to the `dhryansi` directory in the *`install_directory`*`\Examples` directory and Shift-click on `dhry_1.c` and `dhry_2.c` to select them (Figure 3-4).

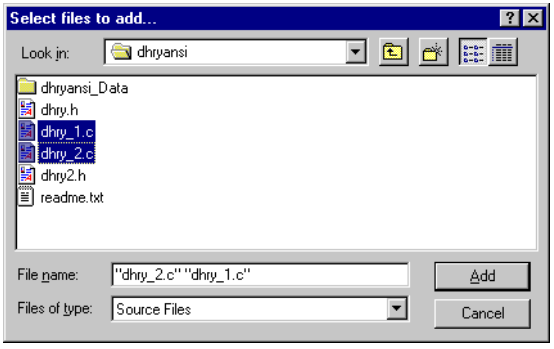

Figure 3-4 Select files to add… dialog

4. Click **Add**. The CodeWarrior IDE displays an Add Files dialog (Figure 3-5 on page 3-7). The dialog contains a checkbox for each build target defined in the current project. In this example, the dialog contains three checkboxes corresponding to the three build targets defined in the ARM Executable Image project stationery.

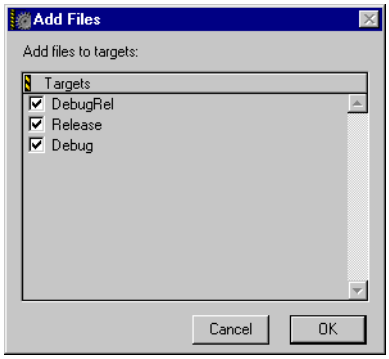

Figure 3-5 Add Files

5. Leave all the build target checkboxes selected and click **OK**. The CodeWarrior IDE adds the source files to each target in the project and displays a Project messages window to inform you that the directory containing the source files has been added to the access paths for each build target (Figure 3-6).

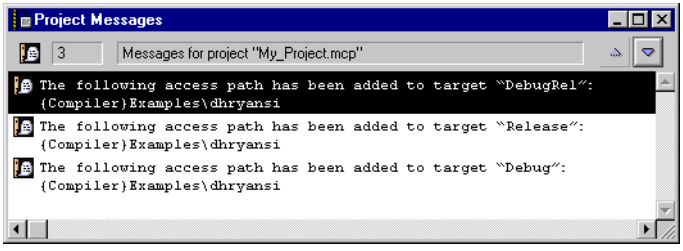

Figure 3-6 Project messages window

The access paths for each build target define the directories that will be searched for source and header files. See the *CodeWarrior IDE Guide* for details.

Note

You do not need to explicitly add the header files for the `dhryansi` project because the CodeWarrior IDE will locate them in the newly added access path. However, you can add header files explicitly if you want.

6. Ensure that the **Files** tab is selected in the project window. The project window displays all the source files in the project. (Figure 3-7). See the *CodeWarrior IDE Guide* for more information on what is displayed when you click the **Link Order** tab and the **Targets** tab.

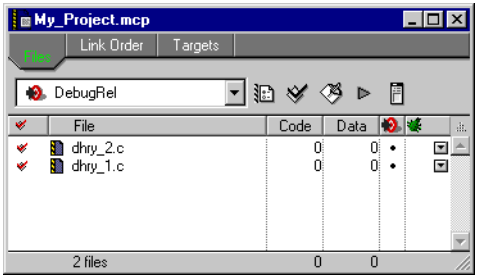

Figure 3-7 Source files in Files view

Configuring the project build targets

This section describes how to configure your example project so that the example `dhryansi` files will compile, and the project build settings are the same as those in the supplied `dhryansi` example project. It describes one way of selecting build targets, and shows how different build target settings can be used in the same project. See the *CodeWarrior IDE Guide* for a complete description of build targets.

Build target settings must be selected separately for each build target in your project. To set build target options for the `dhryansi` example:

1. Ensure that the DebugRel build target is currently selected. By default, the DebugRel build target is selected when you create a new project based on the ARM project stationery. The currently selected build target is displayed in the **Build Target** pop-up menu in the project toolbar (Figure 3-8 on page 3-9).

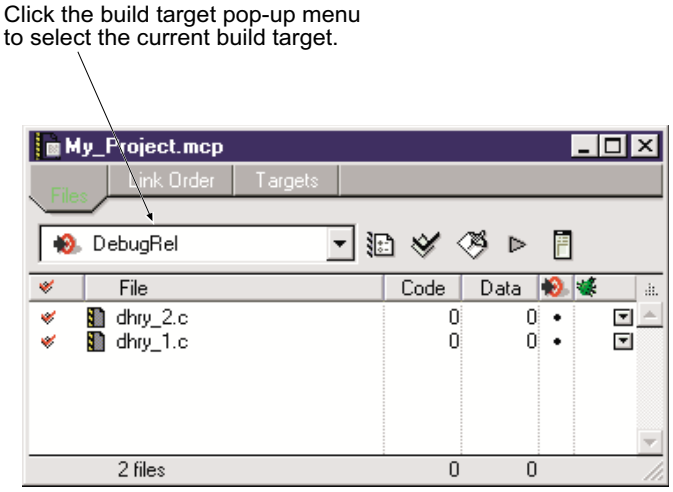

Figure 3-8 Currently selected build target

2. Select **DebugRel Settings…** from the **Edit** menu. The name of this menu item changes depending on the name of the currently selected build target. The CodeWarrior IDE displays the DebugRel Target settings panel (Figure 3-9 on page 3-10). All the target-specific settings are accessible through configuration panels listed at the left of the panel.

Note

Many configuration options are optional, however you should review the target settings for each build target in your project to ensure that they are appropriate for your target hardware, and your development requirements. See the chapter on configuring a build target in the *CodeWarrior IDE Guide* for configuration recommendations.

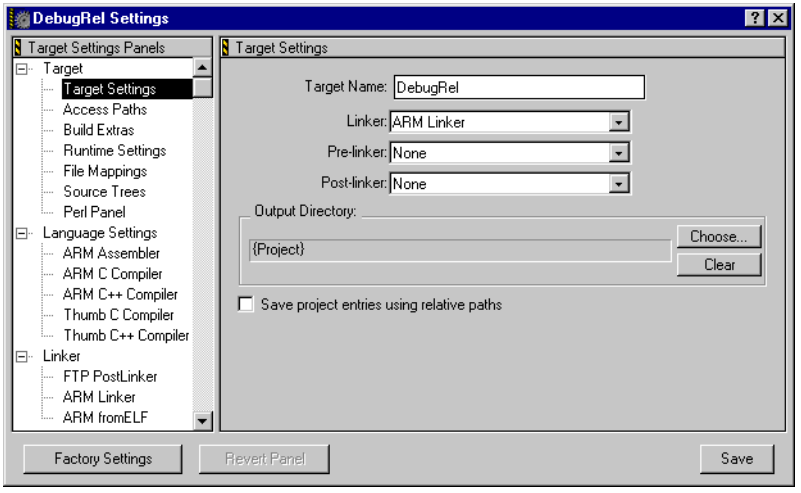

Figure 3-9 DebugRel Settings

3. Click the ARM C compiler entry in the Target Settings Panels list to display the configuration panel for the C compilers. The Target and Source panel is displayed (Figure 3-10). The panel consists of a number of tabbed panes containing groups of configuration options. For this example, the `dhryansi` source requires a predefined macro be set before it will compile.

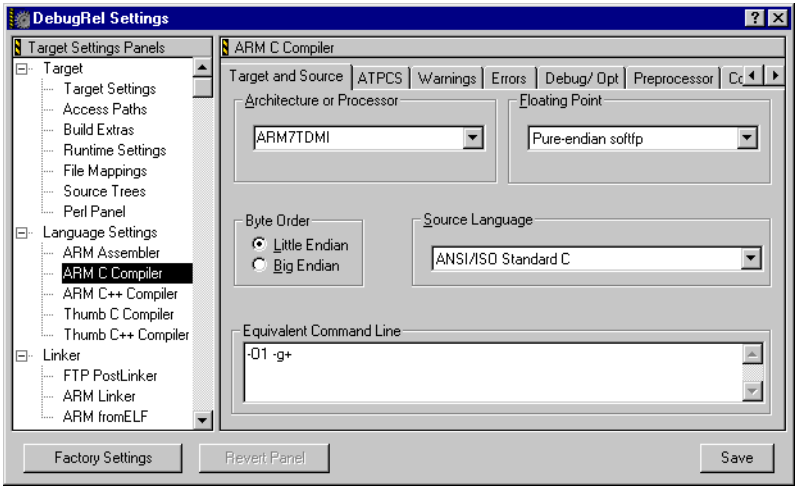

Figure 3-10 ARM C compiler panel

4. Click the Preprocessor tab to display a list of predefined macros (Figure 3-11).

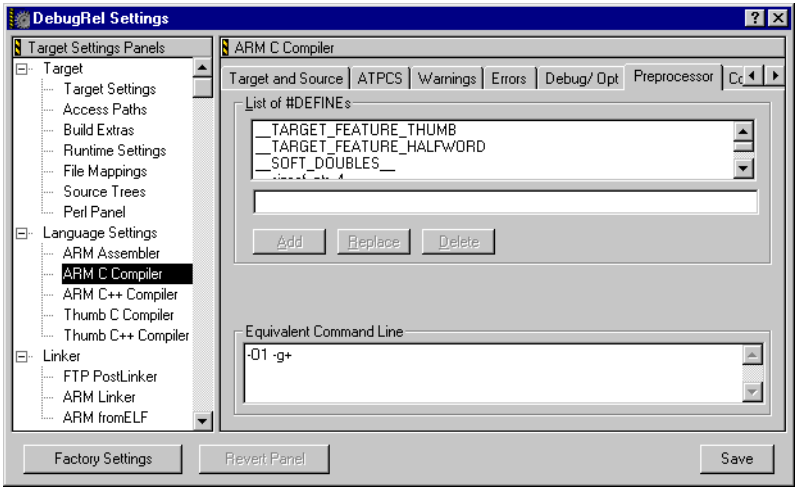

Figure 3-11 ARM C compiler preprocessor pane

5. Type `MSC_CLOCK` into the text field beneath the List of #DEFINES and click **Add** to define the `MSC_CLOCK` macro. The CodeWarrior IDE adds `MSC_CLOCK` to the List of #DEFINES. The Equivalent Command Line text box displays the compiler command-line option required to define `MSC_CLOCK` (Figure 3-12).

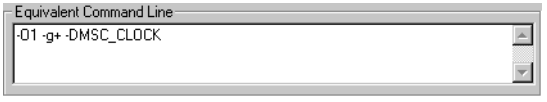

Figure 3-12 MSC_CLOCK defined

6. Click **Save** to save your changes and close the DebugRel Settings panel.

At this point you have defined the `MSC_CLOCK` macro for the DebugRel build target only. You must also define the `MSC_CLOCK` macro for the Release and Debug build targets if you want to use them. To select the Release build target:

1. Ensure that the Project window is currently active.
2. Click the Current Target pop-up menu to display the list of defined build targets (see Figure 3-8 on page 3-9).
3. Select Release from the list of build targets to change the current build target.

4. Apply the steps you followed above to define `MSC_CLOCK` the Release build target.

   Note

   You can also cut and paste build target settings into the Command Line Equivalent text box. Press the Enter key to set the options and update the panel controls. Be careful not to copy command-line options that are inappropriate, such as the optimization and debug settings, from one build target to another.

   Leave the Release Target settings panel open after you have saved your changes.

5. Click on the **Debug/Opt** tab to display Debug and Optimization options for the Release build target (Figure 3-13).

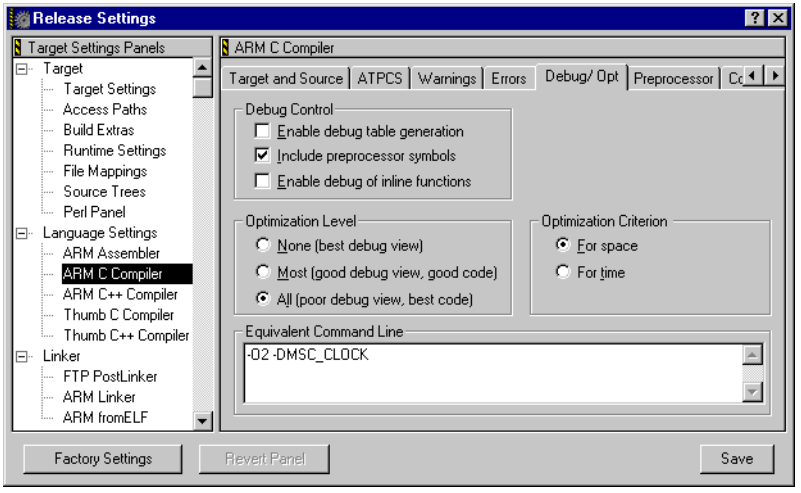

Figure 3-13 Debug/Opt configuration panel

6. Select the **For time** Optimization Criterion button. The Equivalent Command Line text box reflects the change.

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

8. Define `MSC_CLOCK` in the Debug build target in the same way as you have for the DebugRel and Release build targets.

   Your project is now equivalent to the `dhryansi` example project supplied with the ARM Developer Suite.

Note

This example has shown how to use the configuration dialogs to set options for individual build targets. There are configuration panels available for most of the ADS toolchain, including the linker, fromELF, and the assembler. You can use the configuration panels to specify most options available in the tools, including:

- procedure call options
- the structure of output images
- the linker and postlinker to use.
- the ARM debugger to call from the CodeWarrior IDE.

See the chapter on configuring a build target in the *CodeWarrior IDE Guide* for a complete description of build target options.

Building the project

The **Project** menu contains a number of commands to compiler, or compile and link your project files. These commands apply only to the current build target. To compile and link the example project:

1. Ensure that the project window is the currently active window.
2. Select the build target you want to build (see Figure 3-8 on page 3-9). For this example, select the DebugRel build target.
3. Select **Make** from the **Project** menu. The CodeWarrior IDE builds the project by:
   - compiling newly added, modified, and touched source files to produce ELF object files
   - linking object files and libraries to produce an ELF image file, or a partially linked object
   - performing any postlink operations that you have defined for your build target, such as calling fromELF to convert an ELF image file to another format.

     Note

     In the `dhryansi` example there is no postlink operation.

   If the project has already been compiled using a command such as **Bring Up To Date** or **Compile**, the **Make** command performs only the link and postlink steps.

   The compiler displays build information, errors, and warning for the build in a messages window.

Debugging the project

By default, the ARM project stationery is configured to call the AXD debugger to debug and run images built from the CodeWarrior IDE. You can configure the debugger to be called using the ARM Debugger configuration panels for each build target. See the *CodeWarrior IDE Guide* for details.

To execute and debug your example project:

1. Ensure that the project window is the currently active window.
2. Select the build target you want to debug. The **Debug** command applies only to the current build target.
3. Select **Debug** from the **Project** menu. The CodeWarrior IDE compiles and links any source files that are not up to date, and calls the AXD debugger to load and execute the image. See *Debugging the application with AXD* on page 3-31 for more information on using AXD.

3.2 Building from the command line

This section describes how to build an application from the command line. From the command line, you can access:

- the compilers
- the assembler
- the linker
- the CodeWarrior IDE.

3.2.1 Using the compilers from the command line

There are four compiler variants as shown in Table 3-1:

Table 3-1 Compiler variants

Compiler name	Compiler variant	Source language	Compiler output
armcc	C	C	32-bit ARM code
tcc	C	C	16-bit Thumb code
armcpp	C++	C or C++	32-bit ARM code
tcpp	C++	C or C++	16-bit Thumb code

Compiler syntax

The command for invoking the ARM compilers is:

```
compiler [PCS-options] [source-language] [search-paths]
[preprocessor-options] [output-format] [target-options]
[debug-options] [code-generation-options] [warning-options]
[additional-checks] [error-options] [source]
```

Refer to the *ADS Tools Guide* and *ADS Developer Guide* for more information.

Building an example

Sample C source code for a simple application is in *install_directory*\Examples\rom\embed\main.c.

To build the example from the command line:

1. Compile the C file `main.c` with the following command with either:

   `armcc -g -O1 -c main.c` (for ARM)

   `tcc -g -O1 -c main.c` (for Thumb)

   where:

   `-g` Tells the compiler to add debug tables.

   `-O1` Tells the compiler select minimum optimization.

   `-c` Tells the compiler to compile only (not to link).

2. Link the image using the following command:

   `armlink main.o -o embed.axf`

   where:

   `-o` Specifies the output file as `embed.axf`.

3. Use ARMulator to test the image or download the image to a development board and use Multi-ICE.

For more information on using assembly language, C, C++ and the linker to create applications, see the *ADS Developer Guide* and *ADS Tools Guide*.

3.2.2 Using the CodeWarrior IDE from the command line

In some cases you might not require the Graphical User Interface of the CodeWarrior IDE, for example, when a project is part of a larger system that needs to be built automatically without user interaction.

`CMDIDE.EXE` is a console window program that can be started from the command line to build project files that have been created and edited with the CodeWarrior IDE. `CMDIDE.EXE` invokes the CodeWarrior IDE, passes the proper parameters to produce a build, and waits for the IDE to finish its operation.

The command-line arguments are:

`Projectname`
: Specifies the project to use

`/t`*`Targetname`*
: Specifies a target to become the current target

`/r`
: Removes the objects of the current target before building

`/b`
: Builds the current target

`/c`
: Closes the project after building

`/q`
: Quits the IDE after building

`/v[y|n|a]`
: Option for converting projects on open:

  `y` Convert without asking

  `n` Do not convert

  `a` Ask whether to convert.

`/s`
: Forces the command line to be processed in a new instance of the IDE (rather than using a current instance).

If more than one project document is specified to be opened in the command line, the `/t` target and `/b` build command flags apply to the first project document found in the list of documents. If no project is specified in the command line, it will use the usual logic in the IDE (front project or default project) to select the project to build.

For example, to build the `dhryansi` project change directory to the `dhryansi` example directory and type:

```
cmdide dhryansi.mcp /t DebugRel /c /b
```

If no target is specified it uses whatever the current target is for that project.

All build commands are executed in a different process from the one launched from the command line. The original process will return as soon as the command line has been completely processed and the build has completed.

The following codes are returned and can be tested using the `IF ERRORLEVEL` instruction in a batch file:

`0`	No error
`1`	Error opening file
`2`	Project is not open
`3`	IDE is already building
`4`	Invalid target name (for `/t` flag)
`5`	Error changing current target
`6`	Error removing objects
`7`	Build was cancelled
`8`	Build failed
`9`	Process aborted.

Note

Though `IDE.EXE` understands the same parameters as `CMDIDE.EXE`, it is particularly important on Windows 95 or Windows 98 to use `CMDIDE.EXE` to ensure that builds are correctly serialized rather than executed all at once.

3.2.3 Debugging from the command line

You can use the ARM symbolic debugger (armsd) to debug applications from the command line.

AXD can also be driven from the command line. This is useful for batch testing, for example.

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

3.2.4 Using the assembler from the command line

The basic syntax to use the ARM assembler (armasm) from the command-line is:

```
armasm -list listingfile inputfile
```

For example, to assemble the code in a file called `myfile.s`, type:

```
armasm -list myfile.lst myfile.s
```

This produces an object file called `myfile.o`, and a listing file called `myfile.lst`.

For full details of the command-line options and syntax, refer to the Assembler chapter in *ADS Tools Guide*.

Example 3-1 shows a small ARM assembly language program. You can use it to explore the use of the assembler, linker, and the ARM symbolic debugger.

Example 3-1

```
     AREA     AddReg,CODE,READONLY         ; Name this block of code.
     ENTRY                      ; Mark first instruction to call.
main
     ADR r0, ThumbProg + 1      ; Generate branch target address and set bit 0
                                ; hence arrive at target in Thumb state.
     BX r0                      ; Branch and exchange to ThumbProg.
     CODE16                     ; Subsequent instructions are Thumb code.
ThumbProg
     MOV r2, #2                 ; Load r2 with value 2.
     MOV r3, #3                 ; Load r3 with value 3.
     ADD r2, r2, r3             ; r2 = r2 + r3
     ADR r0, ARMProg            ; Generate branch target address with bit 0 zero.
     BX r0                      ; Branch and exchange to ARMProg.
     CODE32                     ; Subsequent instructions are ARM code.
ARMProg
     MOV r4, #4
     MOV r5, #5
     ADD r4, r4, r5
stop MOV r0, #0x18              ; angel_SWIreason_ReportException
     LDR r1, =0x20026           ; ADP_Stopped_ApplicationExit
     SWI 0x0123456              ; ARM semihosting SWI

     END                        ; Mark end of this file.
```

Building the example

To build the example:

1. Enter the code using any text editor and save the file in your current working directory as `addreg.s`.
2. Type `armasm -list addreg.lst addreg.s` at the command prompt to assemble the source file.
3. Type `armlink addreg.o -o addreg` to link the file.

Running the example in the debugger

To load and run the example in the debugger:

1. Type `armsd addreg` to load the module into the command-line debugger.
2. Type `step` to step through the rest of the program one instruction at a time. After each instruction, you can type `reg` to display the registers.

When the program terminates, to return to the command line, type `quit`.

This example contains both ARM and Thumb code. As you step through the program you can see the T-bit in the *Current Program Status Register* (CPSR) changing between a lowercase t and an uppercase T. This indicates the change between ARM and Thumb state.

For further details on ARM and Thumb assembly language instructions, see the *ARM Architecture Reference Manual*.

For further details on ARM and Thumb assembler pseudo-instructions and directives, see the assembler chapter in *ADS Tools Guide*.

For tutorial information on ARM and Thumb assembly language, see the assembly language chapter in *ADS Developer Guide*.

3.2.5 Setting linker options from the command line

The ARM linker, armlink, enables you to:

- link a collection of objects and libraries into an executable image
- partially link a collection of objects into an object that can be used as input for a future link step
- specify where the code and data will be located in memory

- produce debug and reference information about the linked files.

Objects consist of input sections that contain code, initialized data, or the locations of memory that must be set to zero. Input sections can be *read-only* (RO), *read/write* (RW), or *zero-initialized* (ZI) These attributes are used by armlink to group input sections into bigger building blocks called output sections, regions and images. Output sections are approximately equivalent to ELF segments.

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

Linker input and output

Input to armlink consists of:

- One or more object files in ELF Object Format.
- Optionally, one or more libraries created by armar.

Output from a successful invocation of armlink is one of the following:

- an executable image in ELF executable format
- a partially linked object in ELF object format.

For simple images, ELF executable files contain segments that are approximately equivalent to RO and RW output sections in the image. An ELF executable file also has ELF sections that contain the image output sections.

An executable image in ELF executable format can be converted to other file formats by using the fromELF utility.

Linker syntax

The complete linker command syntax is:

```
armlink [-help] [-vsn] [-partial] [-output file] [-elf]
[-ro-base address] [-ropi] [-rw-base address] [-rwpi] [-split]
[-scatter file] [-debug|-nodebug] [-remove (RO/RW/ZI)|-noremove]
[-entry location ] [-keep section-id] [-first section-id]
[-last section-id] [-libpath pathlist] [-scanlib|-noscanlib]
[-locals|-nolocals] [-info topics] [-map] [-symbols]
[-symdefs file] [-xref] [-xreffrom object(section)]
[-xrefto object(section)] [-errors file] [-list file] [-verbose]
[-via file] [-strict] [-unresolved symbol] [input-file-list]
```

See the linker chapter in the *ADS Tools Guide* for more information on the linker options.

Using linker options to position sections

The following linker options control how sections are arranged in the final image and whether the code and data can be moved to a new location after the application starts:

`-ropi` This option makes the load and execution region containing the RO output section position-independent. If this option is not used the region is marked as absolute.

`-ro-base` *address*
This option sets the execution addresses of the region containing the RO output section at *address*. The default address is `0x8000`

`-rw-base` *address*
This option sets the execution addresses of the region containing the RW output section at *address*. The default address is at the end of the RW section.

`-rwpi` This option makes the load and execution region containing the RW and ZI output section position-independent. If this option is not used the region is marked as absolute. The `-rwpi` option is ignored if `-rw-base` is not also used. Usually each writable input section must be read-write position-independent.

`-split` When used with `-ro-base` or `-rw-base`, this option splits the default load region, that contains the RO and RW output sections, into two load regions.

If you want more control over how the sections are placed in an image, use the `-scatter` option and place the positioning information in a scatter-load file.

Using scatter-load files for a simple image

The command-line options (`-ro-base`, `-rw-base`, `-split`, `-ropi`, and `-rwpi`) create simple image types.

You can create the same image types by using the `-scatter` command-line option and a file containing a corresponding scatter load descriptions .

The simplest image type consists of a single load region in the load view and three execution regions in the execution view. The execution regions are placed contiguous in the memory map.

`-ro-base address` is used to specify the load and execution address of the region containing the RO output section. The scatter load description equivalent to using `-ro-base 0x040000` is:

```
LR_1 0x040000 ; define the load region name as LR_1
              ; region starts at 0x040000
{             ; start of execution region descriptions
    ER_RO +0  ; first execution region is called ER_RO
              ; region starts at end of previous region
              ; since there was no previous region,
              ; address is 0x040000
    {
        *(+RO) ; all RO sections go into this region
               ; they are placed consecutively
    }
    ER_RW +0 ; second execution region is called ER_RW
             ; region starts at end of previous region
             ; address is 0x040000 + size of ER_RO region
    {
        *(+RW) ; all RW sections go into this region
               ; they are placed consecutively
    }
    ER_ZI +0 ; last execution region is called ER_ZI
             ; region starts at end of previous region at
             ; 0x040000 + size of ER_RO + size of ER_RW regions
    {
        *(+ZI) ; All ZI region are created here
               ; they are placed consecutively
    }
}
```

This description creates an image with one load region called `LR_1`, whose load address is `0x040000`.

The image has three execution regions, named `ER_RO`, `ER_RW` and `ER_ZI`, that contain the RO, RW and ZI output sections respectively. The execution address of ER_RO is `0x040000`. All three execution regions are placed contiguously in the memory map by using the `+offset` form of the base-designator for the execution region description. This allows an execution region to be placed immediately following the end of the preceding execution region.

For more information on the linker and scatter-load files, see the *ADS Tools Guide*.

Using a scatter load file for a more complex image

The code in *install_directory*\Examples\rom\embed shows how to create a complex scatter-loading application. This application uses memory remapping to exchange the ROM and RAM regions after the application has started.

The example uses:

- Flash is at 0x04000000. An aliased copy of the FLASH appears at 0x0 on reset
- After remapping, fast SSRAM is at 0x00000000 to hold the exception vectors and the exception handlers
- After remapping, SRAM is at 0x00002000 for the storage of program variables.

The scatter-loading description file shown in Example 3-2 defines one load region (Flash) and three execution regions:

- FLASH (at 0x04000000)
- SSRAM (at 0x00000000)
- SRAM (at 0x00002000).

Example 3-2 scat_d.txt

```
FLASH 0x04000000 0x080000
{
    FLASH 0x04000000 ;
    {
        init.o (Init, +First)
        *(+RO)
    }
    SSRAM 0x0000 ; the code in vectors.o is located at 0x0
    {
        vectors.o (Vect, +First)
    }
    SRAM 0x2000
    {
        * (+RW,+ZI)
    }
}
```

The program code and data is placed in Flash that normally resides at 0x04000000. On reset, an aliased copy of Flash is remapped by hardware to address 0x0. Program execution starts at AREA Init in init.s. The +First option is used to place this code first in the image. After reset the first few instructions of init.s remap 32-bit RAM to address 0x0. The ARM Development (PID) Board remaps its Flash in this way.

Most of the RO code will execute from Flash. The RO execution address is the same as its load address (`0x04000000`), so it does not have to be moved.

SSRAM might be fast 32-bit on-chip RAM. Fast RAM is typically used for the stack and code that must be executed quickly. The exception vectors (`AREA Vect` in `vectors.s`) get relocated from Flash to 32-bit RAM at address `0x0` for speed. The `Vect` code is placed first in the region.

SRAM might be slower 16-bit off-chip DRAM. Slower RAM is typically used for less frequently accessed RW variables and ZI data. The RW data will get relocated from Flash to 16-bit RAM at `0x2000` The ZI data will be created in 16-bit RAM above the RW data.

Example 3-3 illustrates the use of initialization code (`init.s`) to perform ROM/RAM remapping. The portion of the initialization code that handles remapping is also listed.

Example 3-3 ROM/RAM remapping

```
; --- Perform ROM/RAM remapping, if required
IF :DEF: ROM_RAM_REMAP
; On reset, an aliased copy of ROM is at 0x0.
; Continue execution from 'real' ROM rather than aliased copy
    LDR     pc, =Instruct_2
; Remap by writing to ClearResetMap in the RPS Remap and Pause Controller
    MOV     r0, #0
    LDR     r1, =ClearResetMap
    STRB    r0, [r1]
; RAM is now at 0x0.
; The exception vectors (in vectors.s) must be copied from ROM to the RAM
; The copying is done later by the C library code inside __main
ENDIF
```

The initialization code in the C library copies the RO and RW execution regions from their load addresses to their execution addresses before creating any zero-initialized areas.

To build the example, use a batch file or makefile containing the following:

```
armasm -g vectors.s
armasm -g -PD "ROM_RAM_REMAP SETL {TRUE}" init.s
armcc -c -g -O1 main.c -DEMBEDDED -DROM_RAM_REMAP
armcc -c -g -O1 retarget.c
armlink vectors.o init.o main.o retarget.o
        -scatter scat_d.scf -o embed.axf
        -info totals -entry 0x04000000
        -info unused
fromelf embed.axf -bin embed.bin
```

The indented lines are a continuation of the single line above.

This creates:

- an ELF debug image (`embed.axf`) for loading into an ARM debugger
- a binary ROM image (`embed.bin`) suitable for downloading into the Flash memory of the ARM development boards.

3.3 Using ARM libraries

The following runtime libraries are provided to support compiled C and C++:

ANSI C The C libraries consist of:

- The functions defined by the ISO C library standard.
- Target-dependent functions used to implement the C library functions in the semihosted execution environment. You can redefine these functions in your own application.
- Helper functions used by the C and C++ compilers.

C++ The C++ libraries contain the functions defined by the ISO C++ library standard. The C++ library depends on the C library for target-specific support and there are no target dependencies in the C++ library. This library consists of:

- the Rogue Wave Standard C++ Library version 2.0.1
- helper functions for the C++ compiler
- additional C++ functions not supported by the Rogue Wave library.

As supplied, the ANSI C libraries use the standard ARM semihosted environment to provide facilities such as file input/output. This environment is supported by the ARMulator, Angel, Multi-ICE, and EmbeddedICE. You can use the ARM development tools in ADS to develop applications, and then immediately run and debug the applications under the ARMulator or on a development board. See the description of semihosting in the *ADS Debug Target Guide* for more information on the debug environment.

You can re-implement any of the target-dependent functions of the C library as part of your application. This lets you tailor the C library, and therefore the C++ library, to your own execution environment.

The libraries are installed in two subdirectories within `install_directory\lib`:

`armlib` Contains the variants of the ARM C library, the floating-point arithmetic library, and the math library. The accompanying header files are in `install_directory\include`.

`cpplib` Contains the variants of the Rogue Wave C++ library and supporting C++ functions. The Rogue Wave and supporting C++ functions are collectively referred to as the ARM C++ Libraries. The accompanying header files are installed in `install_directory\include`.

Note

- The ARM C libraries are supplied in binary form only.
- The ARM libraries should not be modified. If you want to create a new implementation of a library function, place the new function in an object file, or your own library, and include it when you link the application. Your version of the function will be used instead of the standard library version.
- Normally, only a few functions in the ANSI C library require re-implementation in order to create a target-dependent application.
- The source for the Rogue Wave Standard C++ Library is not freely distributable. It can be obtained from Rogue Wave Software Inc., or through ARM Ltd, for an additional licence fee. See the Rogue Wave online documentation in *install_directory*`\Html` for more about the C++ library.

3.3.1 Using the ARM libraries in a semihosted environment

If you are developing an application that will run in a semihosted environment for debugging, you must have an execution environment that supports the ARM and Thumb semihosting SWIs and has sufficient memory.

The execution environment can be provided by either:

- using the standard semihosting functionality that is present by default in, for example, ARMulator, Angel, and Multi-ICE
- implementing your own SWI handler for the semihosting SWI.

You do not need to write any new functions or include files if you are using the default semihosting functionality of the library.

3.3.2 Using the ARM libraries in a non-semihosted environment

If you do not want to use any semihosting functionality, you must ensure that either no calls are made to any function that uses semihosting or that such functions are replaced by your own non-semihosted functions.

To build an application that does not use semihosting functionality:

1. Create the source files to implement the target-dependent features.
2. Add the `__use_no_semihosting_swi()` guard to the source.
3. Link the new objects with your application.
4. Use the new configuration when creating the target-dependent application.

You must re-implement functions that the C library uses to insulate itself from target dependencies. For example, if you use `printf()` you will need to re-implement `fputc()`. If you do not use the higher level input/output functions like `printf()`, you do not need to re-implement the lower level functions like `fputc()`.

If you are building an application for a different execution environment, you can re-implement the target-dependent functions (functions that use the semihosting SWI or that depend on the target memory map). There are no target-dependent functions in the C++ library. See the chapter on libraries in the *ADS Tools Guide* for more information.

3.3.3 Building an application without the ARM libraries

Creating an application that has a `main()` function causes the C library initialization functions to be included.

If your application does not have a `main()` function, the C library will not be initialized and the following features will not be available in your application:

- software stack checking
- low-level `stdio`
- signal-handling functions, `signal()` and `raise()` in `signal.h`
- `atexit()`
- `alloca()`.

You can create an application that consists of customized startup code instead of the library initialization code and still use many of the library functions. You must either:

- avoid functions that require initialization
- provide the initialization and low-level support functions.

These applications will not automatically use the full C run-time environment provided by the C library. Even though you are creating an application without the library, some helper functions from the library must be included. There are also many library functions that can be made available with only minor re-implementations. See the chapter on libraries in the *ADS Tools Guide* for more information.

3.4 Using your own libraries

The ARM librarian, armar, enables sets of ELF object files to be collected together and maintained in libraries. Such a library can then be passed to armlink in place of several object files. However, linking with an object library file does not necessarily produce the same results as linking with all the object files collected into the object library file. This is because armlink processes the input list and libraries differently:

- each object file in the input list appears in the output unconditionally, although unused areas are eliminated if the armlink `-remove` option is specified
- a member of library file is only included in the output if it is referred to by an object file or a previously processed library file.

To create a new library called `my_lib` and add all the object files in the current directory, type:

```
armar -create my_lib *.o
```

To delete all objects from the library that have a name starting with `sys_`, type:

```
armar -d my_lib sys_*
```

To replace, or add, three objects in the library with the version located in the current directory, type:

```
armar -r my_lib obj1.o obj2.o obj3.o
```

For more information on armar, see the Utilities chapter in the *ADS Tools Guide*.

Note

The ARM libraries should not be modified. If you want to create a new implementation of a library function, place the new function in an object file or your own library. Include your object or library when you link the application. Your version of the function will be used instead of the standard library version.

3.5 Debugging the application with AXD

AXD enables you to run and debug your ARM-targeted image using any of the following debugging systems:

- ARMulator (the default)
- *Basic ARM Ten System* (BATS)
- Multi-ICE
- EmbeddedICE
- Angel debug monitor.

If you are using the CodeWarrior IDE, you can start the debuggers by selecting **`Run`**. You can also start the debuggers directly and load an image to debug.

If you prefer command-line debugging, you can use armsd or control AXD from a command-line interface.

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

3.5.1 Starting AXD

Start AXD in any of the following ways:

- If you are running under UNIX, either:
  - from any directory type the full path and name of the debugger, for example, `/opt/arm/axd`
  - change to the directory containing the debugger and type its name, for example, `./axd`
- If you are working in the CodeWarrior IDE, open a project and select **Edit** → ***target*** **Settings...** → **Debugger** → **ARM Debugger** to ensure that AXD is the default debugger and other settings are as you require, then click the **Run/Debug** button or select **Debug** from the **Project** menu.
- If you are running Windows 95 or Windows 98, click on the **AXD Debugger** icon in the **ARM Developer Suite v1.0** program folder or select **Start** → **Programs** → **ARM Developer Suite 1.0** → **AXD Debugger**.
- If you are running Windows NT4, double-click on the **axd.exe** icon in the **ARM Developer Suite\Bin** Program group or select **Start** → **Programs** → **ARM Developer Suite v1.0** → **AXD Debugger**.

- If you are using DOS, launch AXD with arguments. The possible arguments (which must be in lower case) for AXD are:

  `-debug` *`ImageName`*

  Load *`ImageName`* for debugging.

  `-exec` *`ImageName`*

  Load and run *`ImageName`*.

  `-logo` Show splash screen (this is the default).

  `-nologo` Suppress splash screen.

  For example, to launch AXD and load `dhryansi.axf` for debugging, type:

  ```
  axd -debug dhryansi.axf
  ```

Loading an image

If you start AXD from the CodeWarrior IDE or use the appropriate parameters on the DOS command line, an image is already loaded into AXD. If you start AXD without specifying an image, use **File→Load Image** to load a new image (Figure 3-14).

Figure 3-14 Loading an image

Stepping through an application

Use **Execute→Step** to step through the application (Figure 3-15).

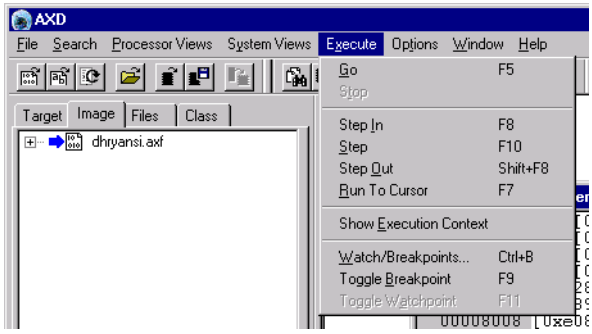

Figure 3-15 The Execute menu

The disassembled code is displayed and a pointer indicates the current position (Figure 3-16). Use **Step** (F10) to execute the next instruction.

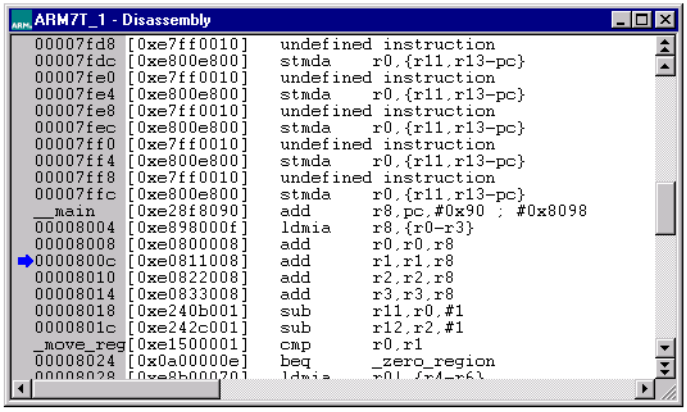

Figure 3-16 Code

Processor view

Use the **Processor Views** menu to monitor the program data during the debug (Figure 3-17).

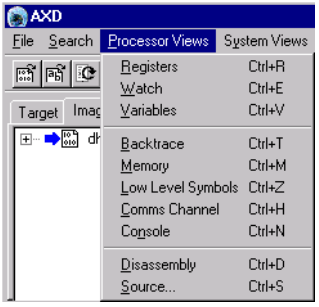

Figure 3-17 Processor Views menu

For example, use **Processor Views→Register** to display a dialog showing the register contents (Figure 3-18).

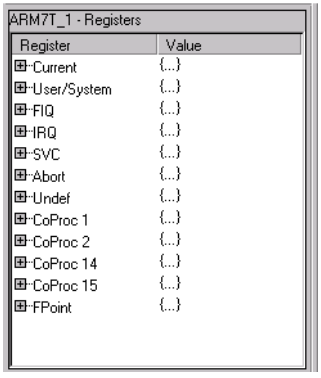

Figure 3-18 Viewing register contents

3.5.2 Configuring ARMulator for AXD

If you install ADS and run AXD, an ARMulator debugging session starts by default, with ARMulator configured by settings held in a default configuration file.

For information on reconfiguring ARMulator, returning to ARMulator after using another debug target, and selecting and configuring other debug targets, refer to the *ADS Debuggers Guide*.

Chapter 4
FLEXlm License Manager

You need a license file to run ADS 1.0 components. This chapter describes how to use the FLEXlm license manager to install a license file to your Windows or UNIX workstation:

4.1 Installing a single node-locked license on a Windows PC

A node-locked license is a license for running ADS on a single PC. All software is installed on the machine and the software does not have to contact a separate license server to validate the license.

4.1.1 Installing a temporary license

Install a temporary license to get you working quickly while you are waiting for the full license:

1. Select **License Installation Wizard** from ARM Developer Suite in the program menu. The help file for the license wizard is displayed (Figure 4-1).

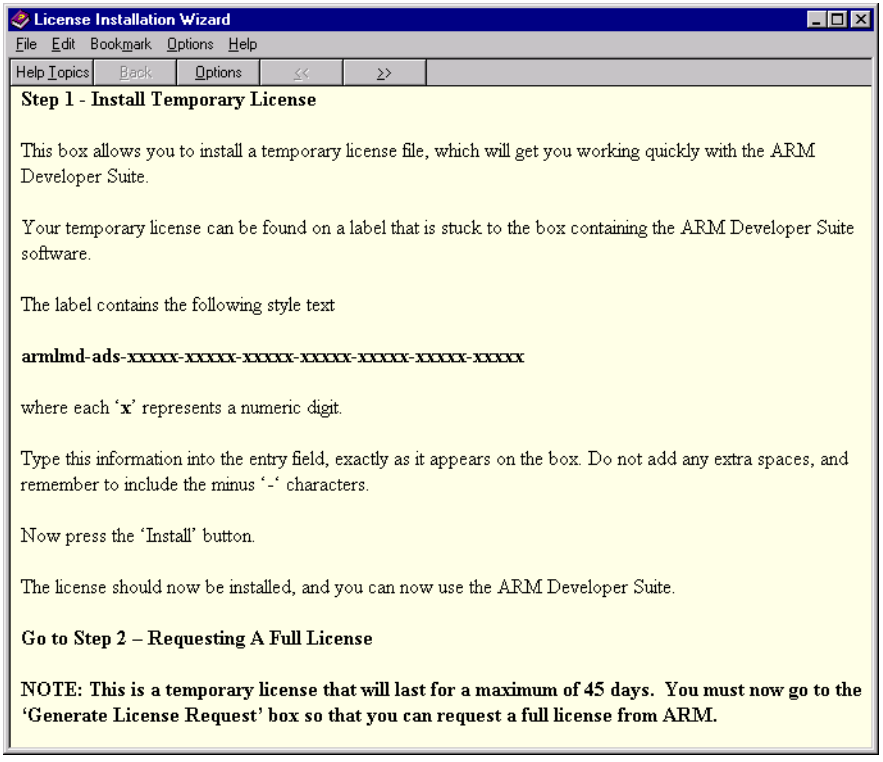

Figure 4-1 License Wizard help

2. The dialog in Figure 4-2 is displayed that allows you to install a license file.

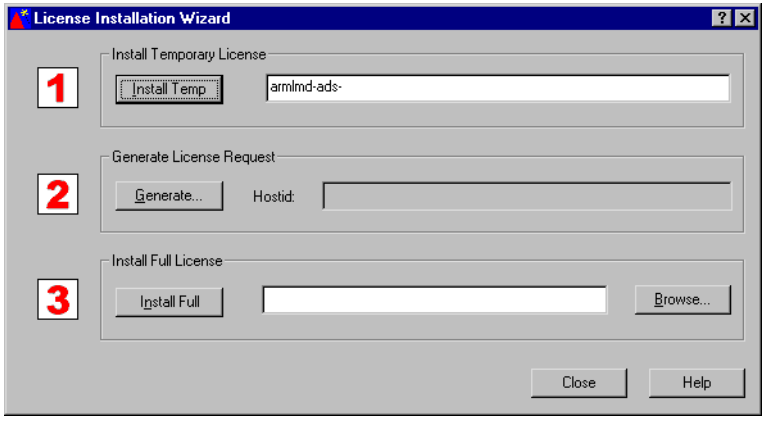

Figure 4-2 License Installation Wizard

3. Your temporary license can be found on a label that is stuck to the box containing the ARM Developer Suite software. The label contains the following style text

   `armlmd-ads-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx`

   where each x represents a numeric digit.

4. Type this information into the entry field, located next to the **Install Temp** button, exactly as it appears on the box.

   Do not add any extra spaces, and remember to include the minus – characters.

5. Click the **Install Temp** button.

6. Click the **Close** button.

7. The license should now be installed, and you can now use the ARM Developer Suite while you are waiting for your full license.

This is a temporary license that will last for a maximum of 45 days. You must follow the instructions in *Generating a license request for a node locked license* on page 4-4 to get a full license from ARM.

4.1.2 Generating a license request for a node locked license

Use the Generate License Request box to gather the information required by ARM for a full license file:

1. Select **License Installation Wizard** from ARM Developer Suite in the program menu. A dialog is displayed that allows you to install a license file.
2. Click the **Generate** button. A new dialog box will appear (Figure 4-3).

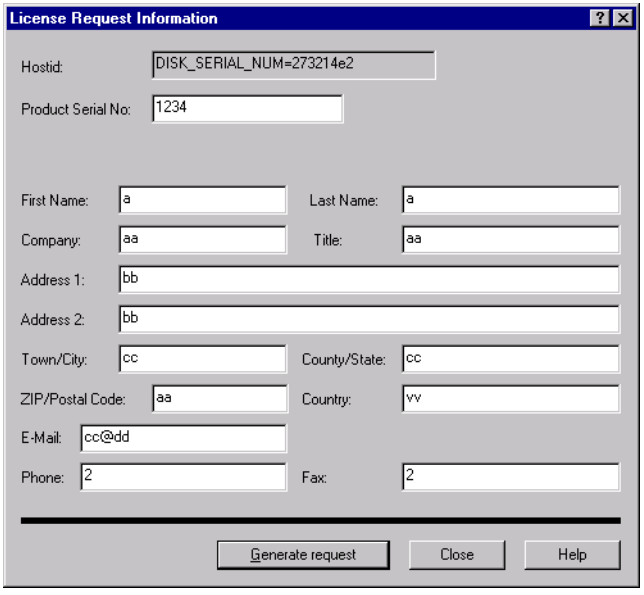

Figure 4-3 Generate request

3. Enter all of your details into the fields on this new window then click the **Generate Request** button.

   The License Wizard will produce a file called `license-request.txt` in the top-level directory where you installed the ARM Developer Suite.
4. E-mail or fax this file to one of the addresses listed within the file.Your full license file should be returned to you within two working days.

   Please ensure that your E-mail program sends this file as ASCII text, not as HTML.
5. Click the **Close** button.

4.1.3 Installing a permanent node locked license

When you receive your license file, you must run the license installation wizard to install the full license:

1. Save the license file to disk as an ASCII text file. It does not matter what you name the file. This file will typically look something like:

```
PACKAGE ads armlmd 1.0 E6A74AF344C6 COMPONENTS="armasm \
        compiler bats armulate axd adwu"
INCREMENT ads armlmd 1.000 permanent uncounted DBF6C33DB1 \
         HOSTID=00105a733bd0 PLATFORMS=i86_n ISSUER= \
         "ARM Limited"
         ck=117

# NOTE : This file should be copied into the "licenses"
# sub-directory of each ADS installation.
#
# Generated for: User Name
# Company: Company Name
# Date: Fri Jan 14 11:36:27 2000
```

2. Select **License Installation Wizard** from ARM Developer Suite in the program menu. A License Installation Wizard dialog is displayed (see Figure 4-2 on page 4-3).

3. Identify the path and file name of the license file by either:
   - entering the name of the disk file into the Install Full entry field.
   - clicking the **Browse** button and browse until you find the file on disk then click the **Open** button to copy the file name into the Install Full entry field (Figure 4-4).

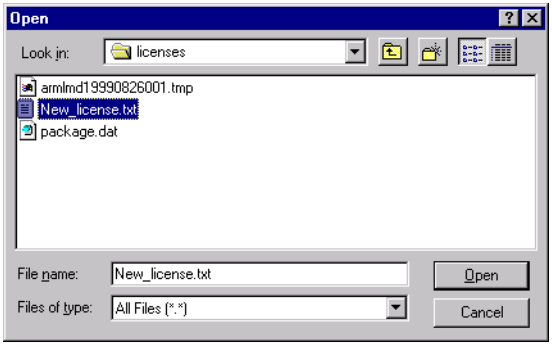

Figure 4-4 Open license file

4. Click the **Install Full** button. The license wizard takes the full license file and installs it so that you can use the product. You will be prompted to delete any temporary, or expired license files from your license directory.

5. Click the **Close** button.

The license wizard saves all the license files it installs using the following naming convention:

```
armlmdYYYYMMDD-SSS.lic
```

where:

`armlmd` Is the name of the ARM vendor daemon.

`YYYY` Is the year.

`MM` Is the month.

`DD` Is the day.

`SSS` Is the sequence number.

For example, the first license file installed on December 16 1999 is saved in the file `armlmd19991216-001.lic`.

Note

FLEXlm requires a hostid to identify specific machines that are licensed to run ADS. For node-locked licenses, the license wizard uses the following as the hostid for the license file requested for your machine:

- the network card ID of your machine, if a network card is installed
- the hard disk serial number, if you do not have a network card installed.

This means that:

- for machines with a network card, you must use the license wizard to request a new license file for your machine if you change the network card
- for machines without a network card, you must use the license wizard to request a new license file for the machine if you reformat your drive c: partition, or change the primary hard disk.

For a floating license, the hostid is determined from the network card for the server.

4.2 Installing a floating license for a Windows client

A floating license is a license that allows any user on the same local network as the license server to run ADS. The license server counts how many concurrent users of the software there are. When a user on the network uses the software, one of the floating licenses is allocated to the user. The ADS software must be installed on each PC and the local license files must identify the location of the license server. You can have more clients with ADS installed than you have licenses, but the number of concurrent users is limited by the license count.

4.2.1 Installing a temporary license

You can install a temporary license to get you working quickly while you are waiting for the system administrator to install the full license on the server:

1. Select **License Installation Wizard** from ARM Developer Suite in the program menu. The help file for the license wizard is displayed.
2. A dialog is displayed that allows you to install a license file.
3. Your temporary license can be found on a label that is stuck to the box containing the ARM Developer Suite software. The label contains the following style text

   `armlmd-ads-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx`

   where each x represents a numeric digit.
4. Type this information into the entry field, located next to the **Install Temp** button, exactly as it appears on the box.

   Do not add any extra spaces, and remember to include the minus – characters.
5. Click the **Install Temp** button.
6. Click the **Close** button.
7. The license should now be installed, and you can now use ADS while you are waiting for your full license. This is a temporary license that will last for a maximum of 45 days.
8. Contact your system administrator and ask for a permanent license to be installed on the license server. See *Configuring the license server* on page 4-11 for information on server licenses.

4.2.2 Requesting a full floating license

Floating licenses require that a machine is set up to run the license server software. The task of requesting floating licenses and setting up the license server is usually handled by system administration. In this case, contact you system administrator.

If you are responsible for system administration, or are responsible for administering your own FLEXlm license management, refer to *Configuring the license server* on page 4-11 for information on configuring license servers for ADS.

4.2.3 Installing a permanent license

After the license file has been installed on the license server, use the License Installation Wizard to install the same license file on all ADS client installations:

1. Select **License Installation Wizard** from ARM Developer Suite in the program menu. A dialog is displayed that allows you to install a license file.
2. Identify the path and file name of the license file by either:
   - enter the name of the server directory and disk file into the entry field next to the **Install Full** button.
   - click the **Browse** button and browse until you find the file on the server then click the **Open** button to copy the file name into the entry field.
3. Click the **Install Full** button. The license wizard will now take the location of the full license file so that you can use the product.

   The server that has the full license installed must be accessible when you run ADS on your client PC.
4. Click the **Close** button.

4.3 Installing a floating license for a UNIX client

A floating license is a license that allows any user on the same local network as the license server to run ADS. The license server counts how many concurrent users of the software there are. When a user on the network uses the software, one of the floating licenses is allocated to the user. The ADS software must be installed on the UNIX workstation and the local license files must identify the location of the license server. You can have more clients with ADS installed than you have licenses, but the number of concurrent users is limited by the license count.

4.3.1 Installing a temporary license (UNIX)

You can install a temporary license to get you working quickly while you are waiting for the system administrator to install the full license on the server:

1. Install ADS on the client machine.
2. Change into the `$ARMHOME/licenses` directory.
3. Run the install utility by typing:
   ```
   lmutil lminstall
   ```
4. When asked to `"Enter path to output license file"`, press **Return** to accept the default filename. The default filename is based on the current date.
5. At the : prompt, enter your temporary license number. The license number looks similar to:
   ```
   armlmd-ads-05876-54321-21098-76543-21098-76543-32109-9876
   ```
6. Press **Return**. The message `Good` should be displayed.
7. Press **Return** twice. The data has been converted into a license file.
8. You should now be able to use the license-managed software.
9. Follow the instructions in *Using a permanent license on a remote server* on page 4-10 to obtain a permanent license.

This is a temporary license that will last for a maximum of 45 days. You must follow the instructions in *Generating a license request for a node locked license* on page 4-4 to get a full license from ARM.

4.3.2 Using a permanent license on a remote server

When the permanent license file is installed on the license server, you must copy the file to each client machine. The client machine uses the license file to identify the location of the license server.

Using the default location for the license file

The license-managed software searches `$ARMHOME/licenses/` for the license files. All files in this directory with an extension of `.lic` are examined for licenses. (`$ARMHOME` is the top-level directory where the software was installed.) Any existing temporary licenses should be removed from this directory when you install the full license.

If the license file is copied into the default location on each client machine, no additional steps are required to enable the client to run the software.

Changing the location of the license file

The default address can be overridden by setting the environment variable `ARMLMD_LICENSE_FILE` to an appropriate value. The way you identify the location depends on the machine holding the full license and how that machine is reached from the client.

4.3.3 Using a permanent license on the client workstation

If you are installing a single floating license, you might want ADS and the license server to be on the same workstation. The workstation is then managing itself. You must either copy the license file to the default location, or identify it using the `ARMLMD_LICENSE_FILE` environment variable.

4.4 Configuring the license server

Licensing is controlled by a license file that describes the software you can use and how many copies of it you can run concurrently.

You must specify one or more computers to act as a license server, on which license management software runs. Any computer running FLEXlm licensed software must either be a license server or have access to a license server.

The license server can be any one of:

- your local machine where the ADS software is installed
- a remote machine
- several remote machines.

If you choose to use more than one, you must use three license server machines. These communicate with one another, and co-ordinate the licensing. The advantage of this is that if one of the license server machines fails to operate correctly the other two will continue to allow licensed software to be used. This arrangement is known as a *3-server redundant set*.

Remote license servers do not need to be running on the same hardware platform as the software they are controlling. For example, a Solaris UNIX machine can act as the license server for PC clients.

You must install the FLEXlm software on a license server. See *Installing FLEXlm software on the server* on page 4-12.

Once you have installed the software, you can modify your license and use the FLEXlm utilities:

- *Requesting and installing floating licenses* on page 4-13
- *Starting the server software* on page 4-16
- *UNIX FLEXlm license management utilities* on page 4-18.

4.4.1 Installing FLEXlm software on the server

License management software for various platforms is supplied on the CD-ROM of any ARM license-managed software.

Installing the license software on a Windows server

For Windows 95/98 and Windows NT, the software is located in `flexlm\win32`.

Note

Use of a Windows 95/98 machine as a license server is strongly discouraged.

Before applying for a license file you must install the FLEXlm license management software, as follows:

1. Copy `flexlm.cpl` into your `system` directory. This is typically:
   - `c:\winnt\system32` on Windows NT
   - `c:\windows\system` on Windows 95/98.
2. Copy the other files into a directory in your `PATH`.

Installing the license software on UNIX server

The following list shows the minimum versions of UNIX platforms supported, and the subdirectory containing the appropriate software for each:

Solaris 2.5 `flexlm/solaris`

HP-UX 9.x `flexlm/hpux`

SunOS 4.1.4 `flexlm/sunos`

Each directory contains the software in TAR file format, in a file called `flexlm.tar`.

Before applying for a license file you must install the FLEXlm license management software, as follows:

1. Copy the TAR file from the appropriate directory onto each license server machine. The destination directory must be included in your `PATH`.
2. On each license server machine, unTAR the file using the command:

   ```
   tar xvf flexlm.tar
   ```
3. When you have unTARed the software you need to run the `makelinks.sh` script. Change into the directory containing the unTARed software and type:

   ```
   sh makelinks.sh
   ```


4. Follow the instructions in *Requesting and installing floating licenses*, below, to obtain a permanent license.

4.4.2 Requesting and installing floating licenses

This section describes how to obtain and install your floating licenses.

Obtaining your floating licenses

The installation directory contains a license application form in a file called `license_request_form.txt`. Please fill in your details on this form and return it to ARM Limited, either by email or by Fax. A permanent floating license will be sent to you.

What to do with your license file

Make a copy of the license file on each of your license servers, as follows:

- If you receive the license file by email you can either copy the license file section out of the message, or save the entire message to disk. The license server ignores all lines except those that start with `SERVER`, `VENDOR`, `USE_SERVER`, `DAEMON`. `FEATURE`, `INCREMENT`, and `PACKAGE`.
- If you receive the license file by fax you must create a text file and key in the information using an editor of your choice. When data entry is complete, you can use the `lmutil lmcksum` utility to check that you typed everything in correctly. Instructions for using `lmcksum` are given under *UNIX FLEXlm license management utilities* on page 4-18.
- You can save the license file in any directory on each license server. However, it is strongly recommended that this be on a locally mounted file system.
- The license file you receive contains the following line that gives the name of the ARM vendor daemon:

  ```
  VENDOR armlmd
  ```

  FLEXlm searches the following directories to find the ARM vendor daemon:

  1. the current directory
  2. the directories in the `PATH` environment variable
  3. the directory that the license server daemon (`lmgrd`) was started from.

You must add the full pathname to the armlmd program as the second parameter on the VENDOR line. For example:

Windows

```
VENDOR armlmd c:\flexlm\programs\armlmd.exe
```

UNIX

```
VENDOR armlmd /opt/flexlm/tools/armlmd
```

See the enduser.pdf FLEXlm documentation in the flexlm directory of your ADS installation disk for full details of the options you can specify on the VENDOR line of your license file.

See *Customizing your license file* on page 4-14 for more information.

Customizing your license file

Your license file contains information similar to that shown in one of the following examples:

Example 4-1: Typical 1-server license file

```
PACKAGE ads armlmd 1.0 E9AB4AD388E7 COMPONENTS="armasm compiler bats \
    armulate axd adwu"
SERVER jupiter 00107702f746 27000
VENDOR armlmd
USE_SERVER
INCREMENT ads armlmd 1.000 permanent 1 4BD7E161B142FF3 DUP_GROUP=UHD \
    ISSUER="ARM Limited" ck=79
```

Example 4-2: Typical 3-server license file

```
SERVER jupiter 00107702f746 27000
SERVER saturn 80af8111f3 27000
SERVER uranus 8187362243 27000
VENDOR armlmd
USE_SERVER
INCREMENT ads armlmd 1.000 permanent 1 4BD7E161B142FF3 DUP_GROUP=UHD \
    ISSUER="ARM Limited" ck=79
```


Although you must not change INCREMENT lines, you might need to change the SERVER and VENDOR lines in your license file.

You might need to change SERVER and VENDOR lines for the following reasons:

Hostname on SERVER line

You might need to change the hostname of a license server. If you change the hostname, you must change the hostname in all copies of the license file that refer to that server.

If you supplied three hostnames on the license request form, there are three SERVER lines in the license file.

TCP port on SERVER line

It is possible to specify on a SERVER line the TCP port that the license manager uses to communicate with the licensed software. If not specified the license manager will use the next available port in the range 27000-27009. When connecting to a server, an application tries all the ports in the range 27000-27009.

A port number must be specified on each SERVER line if a 3-server license is in use.

Daemon path on VENDOR line

FLEXlm searches the following directories to find the ARM vendor daemon:

1. the current directory
2. the directories in the PATH environment variable
3. the directory that the license server daemon (lmgrd) was started from.

If the armlmd program is not in one of these places, you must edit this line to specify the location of the program. You must add the full pathname to the armlmd program as the second parameter on the VENDOR line. For example:

Windows

```
VENDOR armlmd c:\flexlm\programs\armlmd.exe
```

UNIX

```
VENDOR armlmd /opt/flexlm/tools/armlmd
```

See the enduser.pdf FLEXlm documentation in the flexlm directory of your ADS installation disk for full details of the options you can specify on the VENDOR line of your license file.

Increment lines

Increment lines describe the licenses that are available, and *must not* be altered. If they are altered the license is invalidated and the feature no longer operates.

Each Increment line specifies the feature name, the vendor daemon name, the feature version, the expiration date of the license (a year of 0 means the license never expires), the number of concurrent licenses available, and the license key.

4.4.3 Starting the server software

The license management software on the license server must be running to handle requests for license validation from client machines.

UNIX

To start the license server software on each machine, go to the directory containing the license server software and type:

```
nohup lmgrd -c license_file_name -l logfile_name &
```

where:

license_file_name
: Specifies the fully qualified pathname of the license file

logfile_name
: Specifies the fully qualified pathname to a log file.

When you have started the license server, you can type:

```
cat logfile_name
```

to see the output from the license server.

Windows

To start the license server software on each machine:

1. Open the **Control Panel** on the license server machine.
2. Start the **FLEXlm License Manager** application.
3. Click the **Setup** tab. Fill in the fields on the page that is displayed.
4. Click the **Control** tab.
5. When prompted to save changes, select **Yes**.

6. Click the **Start** button.
7. The license server will start running.

Running your application

Copy the license file into the ADS `licenses` subdirectory of each client machine that will use the license server. This file must be given a `.lic` filename extension. You can now run the license managed software in ADS.

4.4.4 Using FLEXlm with more than one product

FLEXlm is a widely used product for license management, so it is possible that you have more than one product using FLEXlm.

Detailed information on FLEXlm is provided in the file `enduser.pdf` located in the `flexlm` subdirectory of the installation CD.

The latest version of the FLEXlm software will always work with vendor daemons built using previous versions. Consequently you must always use the latest version of `lmgrd` and the FLEXlm utilities.

Note

The FLEXlm software currently shipped by ARM is FLEXlm version 6.1g.

If you have multiple products using FLEXlm you might encounter two situations:

- all the products use the same license server
- all the products use different license servers.

All products use the same server

If the license files for every product contain exactly the same `SERVER` lines, ignoring different TCP port numbers, then there are two possible solutions:

- Start a separate `lmgrd` daemon for each license file. There are no real disadvantages with this approach, as the separate daemons consume very little system resources or CPU time.
- Combine the license files together. Take the `SERVER` line from one of the license files then add all of the other lines, that is the `DAEMON/VENDOR` and `FEATURE/INCREMENT` lines, to create a new license file.

  You must store the new combined license file in

  `/usr/local/flexlm/licenses/license.lic`

or give its location using the LM_LICENSE_FILE environment variable.

Now start lmgrd using the new license file. Remember that you must use the latest version of lmgrd that is used by any of the products. You can use the command lmgrd -v or lmver lmgrd to find out the version of each lmgrd.

If the version of lmgrd is earlier than any of the vendor daemons, you see error reports such as: Vendor daemon cannot talk to lmgrd (invalid data returned from license server)

Leave a symbolic link to the new license file in all the locations that held the original license files.

All products use different license servers

If all the products use different hosts to run the license managers, then you must keep separate license files for each product.

Set the LM_LICENSE_FILE environment variable to point to the locations of all the license files, for example:

```
setenv LM_LICENSE_FILE license_file1:license_file2:
...:license_filen
```

Note

FLEXlm version 6.1 allows each software vendor to have an individual environment variable for finding the license file for their products. The environment variable name is *xxx*_LICENSE_FILE where *xxx* is the name of the vendor license daemon. In the case of software from ARM Limited the vendor daemon is called ARMLMD, therefore the environment variable for ARM software is ARMLMD_LICENSE_FILE. FLEXlm vendor daemons always look for the vendor-specific environment variable, ahead of the LM_LICENSE_FILE environment variable.

4.4.5 UNIX FLEXlm license management utilities

The flexlm directory on your product CD-ROM contains subdirectories holding the license manager utilities and the ARM vendor daemon (armlmd) for various platforms.

Installing FLEXlm software on the server on page 4-12 describes how to install the software on your (one or three) UNIX license server machines.

All the license tools are actually contained within the single executable lmutil, the behavior of the program is determined by the value of its argv[0].

Using the utilities on UNIX

Running the script `makelinks.sh` allows you to execute the utilities using their short names, for example you can type `lmver` instead of `lmutil lmver`.

The `lmdown`, `lmremove`, and `lmreread` commands are privileged. If you started `lmgrd` with the `-p 2` switch then you must be a license administrator to run any of these three utilities.

A license administrator is a member of the UNIX `lmadmin` group or, if that group does not exist, a member of group `0`.

In addition, `lmgrd -x` can disable `lmutil lmdown` and `lmutil lmremove`.

All utilities take the following arguments:

`-v` print version and exit.

`-c` *`license_file`*
operate on a specific license file.

lmcksum

`lmutil lmcksum [-k] [-c` *`license_file_name`*`]`

The `lmutil lmcksum` utility performs a checksum of a license file. Use it to check for data entry errors in your license file. `lmutil lmcksum` prints a line-by-line checksum for the file as well as an overall file checksum. If the license file contains `cksum=nn` attributes, the bad lines are indicated automatically.

This utility is particularly useful if you receive your license by fax and typed the file, because of the possibility of data entry errors.

Use the `-k` switch to force the checksum to be case-sensitive.

By default `lmutil lmcksum` checks the contents of `license.dat` in the current directory. Use the `-c` switch to check a different file.

lmdiag

`lmutil lmdiag [-c` *`license_file_list`*`] [-n] [`*`feature`*`]`

This utility allows you to check for problems when you cannot check out a license.

`-c license_file_list`
: Check file(s) on path. If there is more than one file, use a colon separator.

`-n`
: Run in non-interactive mode.

`feature`
: Diagnose this feature only. If you do not specify a feature, all lines of the license file are checked.

The `lmdiag` program first tries to check the feature. If this fails, the reason for failure is printed.

If the check failed because `lmutil lmdiag` could not connect to the license server then you can run extended connection diagnostics. These diagnostics try to check the validity of the port number in the license file. `lmutil lmdiag` displays the port numbers of all ports that are listening and indicates the ones that are `lmgrd` processes. If `lmutil lmdiag` finds the `armlmd` daemon for the for feature being tested, it displays the correct port number to use in the license file.

lmdown

```
lmutil lmdown [-c license_file_list] [-vendor name] [-q]
```

The program allows you to shut down gracefully all license daemons on all nodes (both `lmgrd` and all vendor daemons).

`-c license_file_list`
: Path to file(s) to be shut down. If more than one file, use a colon separator.

`-vendor name`
: If you specify a vendor name, only that vendor daemon is shut down and `lmgrd` is not shut down.

`-q`
: Do not issue the `Are you sure?` prompt.

You should restrict the execution of `lmutil lmdown` to license administrators, by starting `lmgrd` with the `-p -2` switch, as shutting down the server causes loss of licenses.

To disable `lmdown`, the license administrator can use `lmgrd -x lmdown`.

To stop and restart a single vendor daemon, use `lmutil lmdown -vendor name`, then `lmreread -vendor name`.

lmremove

```
lmutil lmremove [-c license_file_list] feature user host display
```

This utility allows you to remove a single user license for a specific feature. For example, when a user is running the software and the host crashes, the user license is sometimes left checked out and unavailable to other users. `lmremove` frees the license and makes it available to other users.

`-c license_file_name`
: The full pathname of the license file to be used. If this is omitted the `LM_LICENSE_FILE` environment variable is used instead.

feature
: The name of the feature the user has checked out.

user
: The name of the user.

host
: The name of the host the user was logged into.

display
: The name of the display where the user was working.

You can obtain the `user`, `host`, and `display` information from the output of `lmutil lmstat -a`.

If the application is active when its license is removed by `lmremove`, it checks out the license again at the next application heartbeat.

lmhostid

```
lmutil lmhostid
```

This program returns the correct host ID on any computer supported by FLEXlm.

lmreread

```
lmutil lmreread [-vendor name] [-c license_file_list]
```

This utility causes the license daemon to reread the license file, and start any new vendor daemons that have been added. All the existing daemons are signalled to reread the license file to check for any changes in their licensing information.

`-vendor` *`name`*

If you specify a vendor name, only that vendor daemon rereads the license file. If the vendor daemon is not running, `lmgrd` starts it.

To disable `lmutil lmreread`, the license administrator can use `lmgrd -x lmreread`.

`lmutil lmreread` does not cause server host names or port numbers to be reread from the license file. To make any changes to those items effective, you must restart `lmgrd`.

To stop and restart a single vendor daemon, use `lmutil lmdown -vendor` *`name`*, then `lmutil lmreread -vendor` *`name`*.

lmstat

```
lmstat [-a] [-A] [-c license_file_list] [-f [feature]] [-i
[feature]] [-s [server]] [-S [daemon]] [-t value]
```

This utility helps you to monitor the status of all network licensing activities, including:

- the daemons that are running
- users of individual features
- users of features served by specific daemons.

The optional arguments are:

`-a` Displays all information.

`-A` Lists all active licenses.

`-c license_file_list`
Uses all the license files listed.

`-f [feature]`
Lists users of a specific feature.

`-i [feature]`
Prints information about the named feature, or all features if *feature* is omitted.

`-s [server]`
Displays status of server node(s).

`-S [daemon]`
Lists all users and features of a specific daemon.

`-t value` Sets the `lmstat` timeout to *value*.

lmver

```
lmutil lmver [filename]
```

This utility reports the FLEXlm version of a specific library or binary file.

4.5 Frequently asked questions about licensing

Q *Why can I not find the LMHOSTID program?*

A If you are using UNIX, you have not run the `makelinks.sh` script that is in the directory that you unTARed the FLEXlm software into. This script creates a series of links to the `lmutil` program, one of which is for `lmhostid`.

If you are using Windows, this command is not available. Instead, type `lmutil lmhostid`.

Q *How does an ADS application find its license file?*

A ADS applications look in the directory `$ARMHOME\licenses` for any files that have a `.lic` filename extension. These files are searched in an order that is determined by the operating system until a valid license is found.

The value of `$ARMHOME` is set by the installation program to be the top-level directory that contains the ADS software. Under Windows, this value is stored in the registry.

Q *How can I store my license files in a different location?*

A The location of the ADS license files can be overridden by setting the environment variables `$ARMLMD_LICENSE_FILE` or `$LM_LICENSE_FILE`. You can set these environment variables to contain one or more filenames, or directory names. The files and directories are searched in order until a valid license is found. If a directory name is found then each file within the directory that has a .lic extension is searched. The contents of `$ARMLMD_LICENSE_FILE` are searched before the contents of the `$LM_LICENSE_FILE`. `$ARMLMD_LICENSE_FILE` is read only by ARM licensed software. `$LM_LICENSE_FILE` is read by all license managed software.

Q *Do I need to have the license file on each client machine?*

A In general, yes you do. However, if you have a single floating license server you can specify the port and server name of the license server using the `ARMLMD_LICENSE_FILE` environment variable.

For example, set `ARMLMD_LICENSE_FILE` to `7000@licserver1` to specify that the license server is running on the machine `licserver1` and is using port number 7000.

4.6 Information for experienced users of FLEXlm

If you are an experience user of FLEXlm then you will need to know the following information about the ARM implementation of this software.

- The version of FLEXlm used by ARM is 6.1g.
- The ARM-specific environment variable used for specifying the location of license files is called `ARMLMD_LICENSE_FILE`.
- The ARM vendor daemon program is called:

  Windows `armlmd.exe`

  UNIX `armlmd`
- You can run your ADS PC or UNIX licenses on a license server that is running on either platform, however you will not be able to run your UNIX ADS software using PC ADS licenses.
- The default location that is searched for ARM license files is:

  Windows `$ARMHOME\licenses`

  UNIX `$ARMHOME/licenses`

  `ARMHOME` is the top-level directory that ADS was installed into. (Under Windows the value for `ARMHOME` is stored in the registry under `HKEY_LOCAL_MACHINE\Software\ARM Limited\ARM Developer Suite\v1.0`)
- All files with a `.lic` extension in this directory are searched for licenses.

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.

E

F

I

L

O

P

R

S

T

