RealView® Compilation Tools

Version 4.0

Utilities Guide

Copyright © 2008-2010 ARM. All rights reserved.

ARM DUI 0382B (ID101213)

RealView Compilation Tools
Utilities Guide

Copyright © 2008-2010 ARM. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
September 2008	A	Non-Confidential	Release 4.0 for RealView Development Suite
23 January 2009	A	Non-Confidential	Update 1 for RealView Development Suite v4.0
10 December 2010	B	Non-Confidential	Update 2 for RealView Development Suite v4.0

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

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

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

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

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

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

RealView Compilation Tools Utilities Guide

Preface

About this book ... viii
Feedback ... xi

Chapter 1 Introduction

1.1 About the ARM image conversion utility ... 1-2
1.2 About the ARM librarian ... 1-5

Chapter 2 Using fromelf

2.1 Protecting code in images and objects with fromelf ... 2-2
2.2 Processing ELF files in an archive ... 2-5
2.3 Using command-line options ... 2-7
2.4 Command-line options ... 2-11

Chapter 3 Using armar

3.1 Using command-line options ... 3-2
3.2 Command-line options ... 3-5

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Preface

This preface introduces the *RealView Compilation Tools Utilities Guide*. It contains the following sections:

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

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

About this book

This book provides information on the following ARM® utility tools supported by the ARM RealView® Compilation Tools:

- ARM image conversion utility, `fromelf`
- ARM librarian, `armar`.

It gives detailed information on the command-line options for each tool and describes how to invoke them.

Intended audience

This book is written for all developers who are producing applications using RealView Compilation Tools. It assumes that you are an experienced software developer and that you are familiar with the ARM development tools.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the ARM utility programs, `fromelf` and `armar`, provided with RealView Compilation Tools.

Chapter 2 ***Using fromelf***

Read this chapter for a description of the `fromelf` utility and how you can use it to change the image format.

Chapter 3 ***Using armar***

Read this chapter for an explanation of the procedures involved in creating and accessing libraries of objects.

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

Typographical conventions

The following typographical conventions are used in this book:

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

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

`monospace` (with "mono" underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
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.

Further reading

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

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

ARM publications

This book contains information on how to use the ARM utility tools supplied with RealView Compilation Tools. Other publications included in the suite are:

- *RealView Compilation Tools Essentials Guide* (ARM DUI 0202)
- *RealView Compilation Tools Compiler User Guide* (ARM DUI 0205)
- *RealView Compilation Tools Compiler Reference Guide* (ARM DUI 0348)
- *RealView Compilation Tools Linker User Guide* (ARM DUI 0206)
- *RealView Compilation Tools Linker Reference Guide* (ARM DUI 0381)
- *RealView Compilation Tools Libraries and Floating Point Support Guide* (ARM DUI 0349)
- *RealView Compilation Tools Assembler Guide* (ARM DUI 0204)

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

- *RealView Compilation Tools Developer Guide* (ARM DUI 0203).

For full information about the base standard, software interfaces, and standards supported by ARM, see the ARM website, `http://infocenter.arm.com/help/index.jsp`.

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

- *ARM Architecture Reference Manual, ARMv7-A® and ARMv7-R® edition* (ARM DDI 0406)
- *ARMv7-M Architecture Reference Manual* (ARM DDI 0403)
- *ARMv6-M Architecture Reference Manual* (ARM DDI 0419)
- ARM datasheet or technical reference manual for your hardware device.

Feedback

ARM Limited welcomes feedback on both RealView Compilation Tools and the documentation.

Feedback on RealView Compilation Tools

If you have any problems with RealView Compilation Tools, contact your supplier. To help them provide a rapid and useful response, give:

- your name and company
- the serial number of the product
- 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 string of the tools, including the version number and build numbers.

Feedback on this book

If you notice any errors or omissions in this book, send an 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.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Chapter 1
Introduction

This chapter introduces the ARM® utility tools, `fromelf` and `armar`, provided with ARM RealView® Compilation Tools. It includes:

- *About the ARM image conversion utility* on page 1-2
- *About the ARM librarian* on page 1-5.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

1.1 About the ARM image conversion utility

The image conversion utility, `fromelf`, enables you to:

- Process ARM ELF object and image files produced by the compiler, assembler, and linker.
- Process all ELF files in an archive produced by `armar`, and output the processed files into another archive if required.
- Convert ELF images into other formats that can be used by ROM tools or directly loaded into memory. The formats available are:
  - Plain binary
  - Motorola 32-bit S-record
  - Intel Hex-32
  - Byte oriented (Verilog Memory Model) hexadecimal
  - ELF. You can resave as ELF, for example, to remove debug information from an ELF image.
- Protect *Intellectual Property* (IP) in images and objects that are delivered to third parties.
- Display information about the input file, for example, disassembly output or symbol listings, to either `stdout` or a text file.

——— **Note** ———

If your image is produced without debug information, `fromelf` cannot:

- translate the image into other file formats
- produce a meaningful disassembly listing.

———————————

See also:

- *fromelf execution modes*
- *Considerations when using fromelf* on page 1-3.

1.1.1 fromelf execution modes

`fromelf` has the following execution modes:

- ELF mode (`--elf`), to resave a file as ELF
- text mode (`--text`, and others), to output information about an object or image file
- format conversion mode (`--bin`, `--m32`, `--i32`, `--vhx`).

See also

- *--bin* on page 2-12
- *--elf* on page 2-25
- *--i32* on page 2-34
- *--m32* on page 2-42
- *--text* on page 2-55
- *--vhx* on page 2-57.

1.1.2 Considerations when using fromelf

Be aware of the following:

- If you use `fromelf` to convert an ELF image containing multiple load regions to a binary format using any of the `--bin`, `--m32`, `--i32`, or `--vhx` options, `fromelf` creates an output directory named *`destination`* and generates one binary output file for each load region in the input image. `fromelf` places the output files in the *`destination`* directory.

  Note

  For multiple load regions, the name of the first non-empty execution region in the corresponding load region is used for the filename.

  If you convert an ELF image containing multiple load regions using either the `--m32combined` or `--i32combined` option, `fromelf` creates an output directory named *`destination`*, generates one binary output file for all load regions in the input image, and then places the output file in the *`destination`* directory.

  ELF images contain multiple load regions if, for example, they are built with a scatter-loading description file that defines more than one load region.

- When using `fromelf`, you cannot:
  - Change the image structure or addresses, other than altering the base address of Motorola S-record or Intel Hex output with the `--base` option.
  - Change a scatter-loaded ELF image into a non scatter-loaded image in another format. Any structural or addressing information must be provided to the linker at link time.

See also

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *--bin* on page 2-12
- *--elf* on page 2-25
- *--i32* on page 2-34

- *--i32combined* on page 2-34
- *--m32* on page 2-42
- *--m32combined* on page 2-43
- *--vhx* on page 2-57.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

1.2 About the ARM librarian

The ARM librarian, `armar`, enables you to collect and maintain sets of ELF object files in standard format `ar` libraries. You can pass these libraries to the linker in place of several ELF object files.

`armar` can:

- create new libraries
- add files to a library
- replace individual files in a library
- replace all files in a library with specified files in a single operation
- control the placement of files in a library.

`armar` can also display information about a specified library. For example, list all members in a library.

Note

When you create, add, or replace object files in a library, `armar` creates a symbol table by default.

See also:

- Chapter 3 *Using armar*
- Chapter 2 *Getting Started with the ARM Linker* in the *Linker User Guide* for information on how the linker processes its input files
- *Libraries and Floating Point Support Guide*.

1.2.1 Considerations when working with library files

Be aware of the following:

- A library differs from a shared object or *Dynamic Link Library* (DLL) in that:
  - symbols are imported from a shared object or DLL
  - code or data for symbols is extracted from an archive into the file being linked.
- Linking with an object library file might not produce the same results as linking with all the object files collected into the object library file. This is because the linker 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.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

— A member of a library file is only included in the output if it is referred to by an object file or a previously processed library file.

The linker recognizes a collection of ELF files stored in an `ar` format file as a library. The contents of each ELF file form a single member in the library.

Chapter 2
Using fromelf

This chapter describes the ARM® image conversion utility, `fromelf`, provided with ARM RealView® Compilation Tools. It includes:

- *Protecting code in images and objects with fromelf* on page 2-2
- *Processing ELF files in an archive* on page 2-5
- *Using command-line options* on page 2-7
- *Command-line options* on page 2-11.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

2.1 Protecting code in images and objects with fromelf

If you are delivering images and objects to third parties, then you might want to protect the code they contain. To help you to protect this code, fromelf provides the `--strip` option and the `--privacy` option. These options remove or obscure the symbol names in the object or image. The option you choose depends on the how much information you want to remove. The effect of these options is different for object files and images.

Note

You must use `--elf` with these options. Because you have to use `--elf`, you must also use `--output`.

To protect code in image files:

Table 2-1 Effect of fromelf --privacy and --strip options on images files

Option	Local symbols	Section names	Mapping symbols	Build attributes
`fromelf --elf --privacy`	Removes the whole symbol table. Removes the `.comment` section name. This is marked as `[Anonymous Section]` in the `fromelf --text` output. Gives section names a default value. For example, changes code section names to `'.text'`.			
`fromelf --elf --strip=symbols`	Removes whole symbol table. Section names remain the same.			
`fromelf --elf --strip=localsymbols`	Removed	Present	Removed	Present

To protect code in object files:

Table 2-2 Effect of fromelf --privacy and --strip options on object files

Option	Local symbols	Section names	Mapping symbols	Build attributes
`fromelf --elf --privacy`	Removes those local symbols that can be removed without loss of functionality. Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.	Gives section names a default value. For example, changes code section names to `'.text'`	Present	Present
`fromelf --elf --strip=symbols`	Removes those local symbols that can be removed without loss of functionality. Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.	Section names remain the same	Present	Present
`fromelf --elf --strip=localsymbols`	Removes those local symbols that can be removed without loss of functionality. Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.	Section names remain the same	Present	Present

The following example produces a new ELF executable image with the complete symbol table removed and various section names changed, use:

```
fromelf --elf --privacy --output=outfile.axf infile.axf
```

See also:

- *Using command-line options* on page 2-7

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

- *--elf* on page 2-25
- *--output=destination* on page 2-44
- *--privacy* on page 2-45
- *--strip=option[,option,...]* on page 2-53
- the following in the *Linker User Guide*:
  - — *About mapping symbols* on page 4-2
- the following in the *Linker Reference Guide*:
  - — *--list_mapping_symbols, --no_list_mapping_symbols* on page 2-57
  - — *--locals, --no_locals* on page 2-58
  - — *--privacy* on page 2-68.

2.2 Processing ELF files in an archive

You can process all ELF files in an archive, or a subset of those files. The processed files together with any unprocessed files are output to another archive.

The following examples show how to process ELF files in an archive, `test.a`, that contains:

```
bmw.o
bmw1.o
call_c_code.o
newtst.o
shapes.o
strmtst.o
```

Example of processing all files in the archive

This example removes all debug, comments, notes and symbols from all the files in the archive:

```
fromelf --elf --strip=all test.a -o strip_all/
```

This creates an output archive with the name `test.a` in the subdirectory `strip_all`

Example of processing a subset of files in the archive

This example removes all debug, comments, notes and symbols from only the `shapes.o` and the `strmtst.o` files in the archive:

```
fromelf --elf --strip=all test.a(s*.o) -o subset/
```

This creates an output archive with the name `test.a` in the subdirectory `subset`. The archive contains the processed files together with the remaining files that are unprocessed.

The following example processes the `bmw.o`, `bmw1.o`, and `newtst.o` files in the archive:

```
fromelf --elf --strip=all test.a(??w*) -o subset/
```

Example of displaying a disassembled version of files in an archive

This example displays the disassembed version of `call_c_code.o` in the archive:

```
fromelf --disassemble test.a(c*)
```

See also:

- *--disassemble* on page 2-24
- *--elf* on page 2-25
- *input_file* on page 2-38
- *--output=destination* on page 2-44

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

- *--strip=option[,option,...]* on page 2-53.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

2.3 Using command-line options

When specifying command-line options, the following rules apply depending on the type of option:

Single-letter options

All single-letter options are preceded by a single dash `-`.

Keyword options

All keyword options are preceded by a double dash `--`. An = or space character is required between the option and the argument.

For example:

`--diag_style=ide`

`--diag_style ide`.

See also:

- *Invoking fromelf*
- *Ordering command-line options* on page 2-10

2.3.1 Invoking fromelf

The `fromelf` command-line syntax is:

```
fromelf [build-options] [debug-options] [diagnostic-options] [help-options]
[image-content-options] [information-options] [license-option] [output-options]
[privacy-options] [project-template-options] input_file
```

`build-options`

Use the following options to control the output format of build attributes:

- *--decode_build_attributes* on page 2-20
- *--dump_build_attributes* on page 2-24
- *--extract_build_attributes* on page 2-28.

`debug-options`

Use the following options to control debug information in output files:

- *--debugonly* on page 2-19
- *--emit=option[,option,...]* on page 2-25
- *--strip=option[,option,...]* on page 2-53.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

`diagnostic-options`

Use the following options to control diagnostic information in output files:

- *--compare=option[,option,...]* on page 2-17
- *--continue_on_error* on page 2-18
- *--diag_error=tag[,tag,...]* on page 2-22
- *--diag_remark=tag[,tag,...]* on page 2-22
- *--diag_style={arm|ide|gnu}* on page 2-23
- *--diag_suppress=tag[,tag,...]* on page 2-23
- *--diag_warning=tag[,tag,...]* on page 2-24
- *--ignore_section=option[,option,...]* on page 2-34
- *--ignore_symbol=option[,option,...]* on page 2-35
- *--relax_section=option[,option,...]* on page 2-47
- *--relax_symbol=option[,option,...]* on page 2-48
- *--show_cmdline* on page 2-52.

`help-options`

Use the following options to shows the main command-line options and the version number of the compiler:

- *--help* on page 2-32
- *--version_number* on page 2-57
- *--vsn* on page 2-58.

`image-content-options`

Use the following options to control miscellaneous factors affecting the image content:

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *--cad* on page 2-15
- *--cadcombined* on page 2-16
- *--cpu=list* on page 2-18
- *--cpu=name* on page 2-19
- *--device=list* on page 2-21
- *--device=name* on page 2-21
- *--disassemble* on page 2-24
- *--emit=option[,option,...]* on page 2-25
- *--expandarrays* on page 2-28
- *--fieldoffsets* on page 2-29
- *--fpu=list* on page 2-31

- *--fpu=name* on page 2-31
- *--globalize=option[,option,...]* on page 2-31
- *--hide=option[,option,...]* on page 2-32
- *--hide_and_localize=option[,option,...]* on page 2-33
- *--in_place* on page 2-36
- *--interleave=option* on page 2-39
- *--linkview, --no_linkview* on page 2-41
- *--localize=option[,option,...]* on page 2-41
- *--qualify* on page 2-46
- *--rename=option[,option,...]* on page 2-49
- *--select=select_options* on page 2-50
- *--show=option[,option,...]* on page 2-51
- *--show_and_globalize=option[,option,...]* on page 2-51
- *--source_directory=path* on page 2-53
- *--strip=option[,option,...]* on page 2-53
- *--symbolversions, --no_symbolversions* on page 2-55.

`information-options`

- *--info=topic[,topic,...]* on page 2-36
- *--only=section_name* on page 2-43
- *--text* on page 2-55
- *-w* on page 2-58.

`license-option`

To make additional attempts to get a floating license, use *--licretry* on page 2-40.

`output-options`

Use the following options to control the output format:

- *--bin* on page 2-12
- *--bincombined* on page 2-13
- *--bincombined_base=address* on page 2-13
- *--bincombined_padding=size,num* on page 2-14
- *--elf* on page 2-25
- *--i32* on page 2-34
- *--i32combined* on page 2-34
- *--m32* on page 2-42
- *--m32combined* on page 2-43
- *--output=destination* on page 2-44

- *--vhx* on page 2-57
- *--widthxbanks* on page 2-59.

`privacy-options`

Use the following options to protect IP in images and objects that are delivered to third parties:

- *--privacy* on page 2-45
- *--strip=option[,option,...]* on page 2-53.

`project-template-options`

Use the following options to control the use of project templates:

- *--project=filename, --no_project=filename* on page 2-45
- *--reinitialize_workdir* on page 2-47
- *--workdir=directory* on page 2-61.

`input_file` Specifies the ELF file to be processed, and is described in *input_file* on page 2-38.

2.3.2 Ordering command-line options

In general, command-line options can appear in any order. However, the effects of some options depend on how they are combined with other related options.

Where options override other options on the same command line, the options that appear closer to the end of the command-line take precedence. Where an option does not follow this rule, this is noted in the description for that option. Use the `--show_cmdline` option to see how `fromelf` has processed the command line. The commands are shown normalized.

See also

- *--show_cmdline* on page 2-52.

2.4 Command-line options

This section lists the command-line options supported by the image conversion utility in alphabetical order.

Note

You can use special characters to select multiple symbolic names in some `fromelf` arguments:

- the wildcard character * can be used to match any name
- the wildcard character ? can be used to match any single character.

If you are using a special character on Unix platforms, you must enclose the options in quotes to prevent the shell expanding the selection.

For example, enter `'*,~*.*'` instead of `*,~*.*`.

2.4.1 `--base [[`*`object_file`*`::]`*`load_region_ID`*`=]`*`num`*

This option enables you to alter the base address specified for one or more load regions in Motorola S-record and Intel Hex file formats.

Restrictions

You must use one of the output formats `--i32`, `--i32combined`, `--m32`, or `--m32combined` with this option.

Syntax

`--base [[`*`object_file`*`::]`*`load_region_ID`*`=]`*`num`*

Where:

`object_file` is an optional ELF input file.

`load_region_ID`

is an optional load region. This can either be a symbolic name of an execution region belonging to a load region or a zero-based load region number, for example `#0` if referring to the first region.

`num` is either a decimal or hexadecimal value.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Wildcard characters ? and * can be used for symbolic names in *`object_file`* and *`load_region_ID`* arguments. Multiple base addresses can be specified in one `--base` option followed by a comma-separated list of arguments. Table 2-3 shows some examples.

All addresses encoded in the output file start at the base address *`num`*. If you do not specify a `--base` option, the base address is taken from the load region address.

Table 2-3 Examples using `--base`

`--base 0`	decimal value
`--base 0x8000`	hexadecimal value
`--base #0=0`	base address for the first load region
`--base foo.o::*=0`	base address for all load regions in `foo.o`
`--base #0=0,#1=0x8000`	base address for the first and second load regions

See also

- *--i32* on page 2-34
- *--i32combined* on page 2-34
- *--m32* on page 2-42
- *--m32combined* on page 2-43
- *Considerations when using fromelf* on page 1-3.

2.4.2 `--bin`

This option produces plain binary output, one file for each load region. You can split the output from this option into multiple files with the *`memory_config`* option.

Example

To convert an ELF file to a plain binary file (for example `outfile.bin`) use:

```
fromelf --bin --output=outfile.bin infile.axf
```

See also

- *--widthxbanks* on page 2-59
- *Considerations when using fromelf* on page 1-3.

2.4.3 `--bincombined`

This option produces plain binary output. It generates one output file for an image containing multiple load regions. By default, the start address of the first load region in memory is used as the base address. `fromelf` inserts padding between load regions as required to ensure that they are at the correct relative offset from each other. Separating the load regions in this way means that the output file can be loaded into memory and correctly aligned starting at the base address.

Use this option with `--bincombined_base` and `--bincombined_padding` to change the default values for the base address and padding.

See also

- *--bincombined_base=address*
- *--bincombined_padding=size,num* on page 2-14.

2.4.4 `--bincombined_base=`*`address`*

This option enables you to lower the base address used by the `--bincombined` output mode. The output file generated is suitable to be loaded into memory starting at the specified address.

Restrictions

You must use `--bincombined` with this option.

Syntax

`--bincombined=`*`address`*

Where:

`address` The start address where the image is to be loaded:

- if the specified address is lower than the start of the first load region, `fromelf` adds padding at the start of the output file
- if the specified address is higher than the start of the first load region, `fromelf` gives an error.

Default

By default the start address of the first load region in memory is used as the base address.

Example

```
--bincombined --bincombined_base=0x1000
```

See also

- *--bincombined* on page 2-13
- *--bincombined_padding=size,num*.

2.4.5 `--bincombined_padding=`*`size`*`,`*`num`*

This option enables you to specify a different form of padding used by the `--bincombined` output mode.

Restrictions

You must use `--bincombined` with this option.

Syntax

```
--bincombined_padding=size,num
```

Where:

size	is 1, 2, or 4 bytes to define whether it is a byte, halfword, or word.
num	is the value to be used for padding.

Note

`fromelf` expects that 2-byte and 4-byte padding values are specified in the appropriate endianness for the input file. For example, if you are translating a big endian ELF file into binary, the specified padding value is treated as a big endian word or halfword.

Default

The default is `--bincombined_padding=1,0xFF`.

Example

The following examples show how to use `--bincombined_padding`:

```
--bincombined --bincombined_padding=4,0x12345678
```

This example produces plain binary output and fills the space between load regions with copies of the 32-bit word 0x12345678.

`--bincombined --bincombined_padding=2,0x1234`

This example produces plain binary output and fills the space between load regions with copies of the 16-bit halfword 0x1234.

`--bincombined --bincombined_padding=2,0x01`

This example when specified for big endian memory, fills the space between load regions with `0x0100`.

See also

- *--bincombined* on page 2-13
- *--bincombined_base=address* on page 2-13.

2.4.6 `--cad`

This option produces a C array definition or C++ array definition containing binary output. You can use each array definition in the source code of another application. For example, you might want to embed an image in the address space of another application, such as an embedded operating system.

If your image has a single load region, the output is directed to `stdout` by default. To save the output to a file, use the `--output` option together with a filename.

If your image has multiple load regions, then you must also use the `--output` option together with a directory name. Unless you specify a full path name, the path is relative to the current directory. A file is created for each load region in the specified directory. The name of each file is the name of the corresponding execution region.

Use this option with `--output` to generate one output file for each load region in the image.

Example

The following examples show how to use `--cad`:

- To produce an array definition for an image that has a single load region, use:

```
fromelf --cad myimage.axf
unsigned char LR0[] = {
    0x00,0x00,0x00,0xEB,0x28,0x00,0x00,0xEB,0x2C,0x00,0x8F,0xE2,0x00,0x0C,0x90,0xE8,
    0x00,0xA0,0x8A,0xE0,0x00,0xB0,0x8B,0xE0,0x01,0x70,0x4A,0xE2,0x0B,0x00,0x5A,0xE1,
    0x00,0x00,0x00,0x1A,0x20,0x00,0x00,0xEB,0x0F,0x00,0xBA,0xE8,0x18,0xE0,0x4F,0xE2,
    0x01,0x00,0x13,0xE3,0x03,0xF0,0x47,0x10,0x03,0xF0,0xA0,0xE1,0xAC,0x18,0x00,0x00,
    0xBC,0x18,0x00,0x00,0x00,0x30,0xB0,0xE3,0x00,0x40,0xB0,0xE3,0x00,0x50,0xB0,0xE3,
    0x00,0x60,0xB0,0xE3,0x10,0x20,0x52,0xE2,0x78,0x00,0xA1,0x28,0xFC,0xFF,0xFF,0x8A,
    0x82,0x2E,0xB0,0xE1,0x30,0x00,0xA1,0x28,0x00,0x30,0x81,0x45,0x0E,0xF0,0xA0,0xE1,
    0x70,0x00,0x51,0xE3,0x66,0x00,0x00,0x0A,0x64,0x00,0x51,0xE3,0x38,0x00,0x00,0x0A,
```

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

```
    0x00,0x00,0xB0,0xE3,0x0E,0xF0,0xA0,0xE1,0x1F,0x40,0x2D,0xE9,0x00,0x00,0xA0,0xE1,
.
.
.
    0x3A,0x74,0x74,0x00,0x43,0x6F,0x6E,0x73,0x74,0x72,0x75,0x63,0x74,0x65,0x64,0x20,
    0x41,0x20,0x23,0x25,0x64,0x20,0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x00,0x00,
    0x44,0x65,0x73,0x74,0x72,0x6F,0x79,0x65,0x64,0x20,0x41,0x20,0x23,0x25,0x64,0x20,
    0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x0C,0x99,0x00,0x00,0x0C,0x99,0x00,0x00,
    0x50,0x01,0x00,0x00,0x44,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
```

- For an image that has multiple load regions, the following commands create a file for each load region in the directory *root*`\myprojects\multiload\load_regions`:

  ```
  cd root\myprojects\multiload
  fromelf --cad image_multiload.axf --output load_regions
  ```

  If `image_multiload.axf` contains the execution regions `EXEC_ROM` and `RAM`, then the files `EXEC_ROM` and `RAM` are created in the `load_regions` subdirectory.

See also

- *--cadcombined*
- *--output=destination* on page 2-44
- Chapter 5 *Using Scatter-loading Description Files* in the *Linker User Guide*.

2.4.7 `--cadcombined`

This option produces a C array definition or C++ array definition containing binary output. You can use each array definition in the source code of another application. For example, you might want to embed an image in the address space of another application, such as an embedded operating system.

If your image has a single load region, the output is directed to `stdout` by default. To save the output to a file, use the `--output` option together with a filename.

If your image has multiple load regions, then you must also use the `--output` option together with a filename. The file contains multiple load regions.

Example

For an image that has multiple load regions, specify the following commands to create a single file containing array definitions for all load regions:

For an image that has multiple load regions, the following commands create the file `load_regions.c` in the directory *root*`\myprojects\multiload`:

```
cd root\myprojects\multiload
fromelf --cadcombined image_multiload.axf --output load_regions.c
```

See also

- *--cad* on page 2-15
- *--output=destination* on page 2-44
- Chapter 5 *Using Scatter-loading Description Files* in the *Linker User Guide*.

2.4.8 `--compare=option[,option,...]`

This option compares two input files and prints a textual list of the differences. The input files must be the same type, either two ELF files or two library files. Library files are compared member by member and the differences are concatenated in the output.

All differences between the two input files are reported as errors unless specifically downgraded to warnings by using the `--relax_section` option.

Syntax

`--compare=option[,option,...]`

Where *option* is one of:

section_sizes

Compares the size of all sections for each ELF file or ELF member of a library file.

section_sizes::*object_name*

Compares the sizes of all sections in ELF objects with a name matching *object_name*.

section_sizes::*section_name*

Compares the sizes of all sections with a name matching *section_name*.

sections Compares the size and contents of all sections for each ELF file or ELF member of a library file.

sections::*object_name*

Compares the size and contents of all sections in ELF objects with a name matching *object_name*.

sections::*section_name*

Compares the size and contents of all sections with a name matching *section_name*.

function_sizes

Compares the size of all functions for each ELF file or ELF member of a library file.

function_sizes::*object_name*

Compares the size of all functions in ELF objects with a name matching *object_name*.

function_size::*function_name*

Compares the size of all functions with a name matching *function_name*.

global_function_sizes

Compares the size of all global functions for each ELF file or ELF member of a library file.

global_function_sizes::*function_name*

Compares the size of all global functions in ELF objects with a name matching *function_name*.

Wildcard characters ? and * can be used for symbolic names in *object_name* and *section_name* arguments. Multiple options can be specified in one `--compare` option followed by a comma-separated list of arguments.

See also

- *--ignore_section=option[,option,...]* on page 2-34
- *--ignore_symbol=option[,option,...]* on page 2-35
- *--relax_section=option[,option,...]* on page 2-47
- *--relax_symbol=option[,option,...]* on page 2-48.

2.4.9 `--continue_on_error`

This option downgrades all recoverable errors to warnings so that `fromelf` can process broken ELF files.

2.4.10 `--cpu=list`

This option lists the supported architecture and processor names that you can use with `--cpu=`*name*.

See also

- *--cpu=name* on page 2-19.

2.4.11 `--cpu=`*`name`*

This option selects disassembly for a specific ARM processor or architecture. It affects how `fromelf` interprets the instructions it finds in the input files.

Syntax

`--cpu=`*`name`*

Where *`name`* is the name of an ARM processor or architecture.

Example

To select the disassembly for the ARM1176JZF-S™ processor, use:

```
--cpu=ARM1176JZF-S
```

See also

- *--cpu=list* on page 2-18
- *--device=list* on page 2-21
- *--device=name* on page 2-21
- *--disassemble* on page 2-24
- *--info=topic[,topic,...]* on page 2-36
- *--text* on page 2-55.

2.4.12 `--datasymbols`

This option modifies the output information of data sections so that symbol definitions are interleaved.

This option can only be used with `--text -d`.

See also

- *--text* on page 2-55.

2.4.13 `--debugonly`

This option removes the content of any code or data sections. This ensures that the output file contains only the information required for debugging, for example, debug sections, symbol table, and string table. Section headers are retained because they are required to act as targets for symbols.

Restrictions

You must use `--elf` with this option.

See also

- *--elf* on page 2-25.

2.4.14 `--decode_build_attributes`

This option prints the contents of the build attributes section in human-readable form for standard build attributes or raw hexadecimal form for nonstandard build attributes.

Note

The standard build attributes are documented in the *Application Binary Interface for the ARM Architecture*.

Restrictions

This option can only be used in text mode.

Example

The following example shows the output for `--decode_build_attributes`:

```
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 69 bytes

    'aeabi' file build attributes:
    0x000000:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000010:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000020:   41 52 4d 00                                        ARM.
        Tag_CPU_name = "ARM7TDMI"
        Tag_CPU_arch = ARM v4T (=2)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_ABI_PCS_GOT_use = Data are imported directly (=1)
        Tag_ABI_PCS_wchar_t = Size of wchar_t is 2 (=2)
        Tag_ABI_FP_denormal = This code was permitted to require that the sign of a flushed-to-zero
number be preserved in the sign of 0 (=2)
        Tag_ABI_FP_number_model = This code was permitted to use only IEEE 754 format FP numbers (=1)
        Tag_ABI_align8_needed = Code was permitted to depend on the 8-byte alignment of 8-byte data
items (=1)
        Tag_ABI_align8_preserved = Code was required to preserve 8-byte alignment of 8-byte data
objects (=1)
        Tag_ABI_enum_size = Enum values occupy the smallest container big enough to hold all values
(=1)
```

```
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion
preserved (=3)
        Tag_compatibility = 2, "ARM"

    'ARM' file build attributes:
    0x000000:   04 01 12 01                                         ....
```

See also

- *--dump_build_attributes* on page 2-24
- *--extract_build_attributes* on page 2-28
- *Application Binary Interface for the ARM Architecture*.

2.4.15 `--device=list`

This option lists the supported device names that can be used with the `--device=name` option.

See also

- *--device=name*.

2.4.16 `--device=name`

This option enables you to specify a microcontroller or *System-on-Chip* (SoC) device name instead of a CPU name. It has the same format as that supported by the ARM compiler.

Each device has default values for CPU and *Floating-Point Unit* (FPU). However, you can override the FPU from the command line by specifying the `--fpu` option after the `--device` option.

See your device documentation for CPU and FPU implementation details.

Syntax

`--device=name`

where *`name`* is a specific device name.

To get a full list of the available devices, use the `--device=list` option.

See also

- *--cpu=list* on page 2-18
- *--cpu=name* on page 2-19

- *--device=list* on page 2-21
- *--fpu=list* on page 2-31
- *--fpu=name* on page 2-31
- *--device=name* on page 2-44 in the *Compiler Reference Guide*
- *--device=name* on page 2-27 in the *Linker Reference Guide*.

2.4.17 `--diag_error=`*`tag[,tag,...]`*

This option sets diagnostic messages that have a specific tag to error severity.

Syntax

`--diag_error=`*`tag[,tag,...]`*

Where *`tag`* can be:

- a diagnostic message number to set to error severity
- `warning`, to treat all warnings as errors.

See also

- *--diag_remark=tag[,tag,...]*
- *--diag_style={arm|ide|gnu}* on page 2-23
- *--diag_suppress=tag[,tag,...]* on page 2-23
- *--diag_warning=tag[,tag,...]* on page 2-24.

2.4.18 `--diag_remark=`*`tag[,tag,...]`*

This option sets diagnostic messages that have a specific tag to remark severity.

Syntax

`--diag_remark=`*`tag[,tag,...]`*

Where *`tag`* is a comma-separated list of diagnostic message numbers.

See also

- *--diag_error=tag[,tag,...]*
- *--diag_style={arm|ide|gnu}* on page 2-23
- *--diag_suppress=tag[,tag,...]* on page 2-23
- *--diag_warning=tag[,tag,...]* on page 2-24.

2.4.19 `--diag_style={arm|ide|gnu}`

This option specifies the style used to display diagnostic messages.

Syntax

`--diag_style=`*`string`*

Where *`string`* is one of:

`arm`	Display messages using the ARM style.
`ide`	Include the line number and character count for any line that is in error. These values are displayed in parentheses.
`gnu`	Display messages in the format used by GNU.

Default

The default is `--diag_style=arm`.

See also

- *--diag_error=tag[,tag,...]* on page 2-22
- *--diag_remark=tag[,tag,...]* on page 2-22
- *--diag_suppress=tag[,tag,...]*
- *--diag_warning=tag[,tag,...]* on page 2-24.

2.4.20 `--diag_suppress=`*`tag`*`[,`*`tag`*`,...]`

This option disables diagnostic messages that have the specified tags.

Syntax

`--diag_suppress=`*`tag`*`[,`*`tag`*`,...]`

Where:

`tag``[,`*`tag`*`,...]`	is a comma-separated list of diagnostic message numbers specifying the messages to be suppressed.

See also

- *--diag_error=tag[,tag,...]* on page 2-22
- *--diag_remark=tag[,tag,...]* on page 2-22
- *--diag_style={arm|ide|gnu}*

- *--diag_warning=tag[,tag,...]*.

2.4.21 `--diag_warning=tag[,tag,...]`

This option sets diagnostic messages that have a specific tag to warning severity.

Syntax

`--diag_warning=tag[,tag,...]`

Where `tag` can be:

- a diagnostic message number to set to warning severity
- `error`, to downgrade all errors to warnings.

See also

- *--diag_error=tag[,tag,...]* on page 2-22
- *--diag_remark=tag[,tag,...]* on page 2-22
- *--diag_style={arm|ide|gnu}* on page 2-23
- *--diag_suppress=tag[,tag,...]* on page 2-23.

2.4.22 `--disassemble`

This option displays a disassembled version of the image to `stdout`. If you use this option with `--output destination`, you can reassemble the output file with `armasm`.

You can use this option to disassemble either an ELF image or an ELF object file.

Note

The output is not the same as that from `--emit=code` and `--text -c`.

See also

- *--cpu=name* on page 2-19
- *--emit=option[,option,...]* on page 2-25
- *--interleave=option* on page 2-39
- *--output=destination* on page 2-44
- *--text* on page 2-55.

2.4.23 `--dump_build_attributes`

This option prints the contents of the build attributes section in raw hexadecimal form.

Restrictions

This option can only be used in text mode.

Example

The following example shows the output for `--dump_build_attributes`:

```
...
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 69 bytes

    0x000000:   41 33 00 00 00 61 65 61 62 69 00 01 29 00 00 00    A3...aeabi..)...
    0x000010:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000020:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000030:   41 52 4d 00 11 00 00 00 41 52 4d 00 01 09 00 00    ARM.....ARM.....
    0x000040:   00 04 01 12 01                                     .....
```

See also

- *--decode_build_attributes* on page 2-20
- *--emit=option[,option,...]*
- *--extract_build_attributes* on page 2-28
- *--text* on page 2-55.

2.4.24 `--elf`

This option selects ELF output mode.

Use with `--strip=debug,symbols` to remove debug information from an ELF image.

See also

- *--disassemble* on page 2-24
- *--in_place* on page 2-36
- *--output=destination* on page 2-44
- *--strip=option[,option,...]* on page 2-53.

2.4.25 `--emit=`*`option`*`[,`*`option,...`*`]`

This option enables you to specify the elements of an ELF object that you want to appear in the textual output. The output includes ELF header and section information.

Restrictions

This option can only be used in text mode.

Syntax

`--emit=`*`option`*`[,`*`option`*`,...]`

Where *`option`* is one of:

`addresses` This option prints global and static data addresses (including addresses for structure and union contents). It has the same effect as `--text -a`.

This option can only be used on files containing debug information. If no debug information is present, a warning message is generated.

Use the `--select` option to output a subset of the data addresses.

If you want to view the data addresses of arrays, expanded both inside and outside structures, use the `--expandarrays` option with this text category.

`build_attributes`

This option prints the contents of the build attributes section in human-readable form for standard build attributes or raw hexadecimal form for nonstandard build attributes.

`code` This option disassembles code, alongside a dump of the original binary data being disassembled and the addresses of the instructions. It has the same effect as `--text -c`.

Note

Unlike `--disassemble`, the disassembly cannot be input to the ARM assembler.

`data` This option prints contents of the data sections. It has the same effect as `--text -d`.

`data_symbols`

This option modifies the output information of data sections so that symbol definitions are interleaved.

`debug_info` This option prints debug information. It has the same effect as `--text -g`.

`dynamic_segment`

This option prints dynamic segment contents. It has the same effect as `--text -y`.

`exception_tables`

This option decodes exception table information for objects. It has the same effect as `--text -e`.

`frame_directives`

This option prints the contents of `FRAME` directives in disassembled code as specified by the debug information embedded in an object module.

Use this option with `--disassemble`.

`got` This option prints the contents of the *Global Offset Table* (GOT) objects.

`heading_comments`

This option prints heading comments at the beginning of the disassembly containing tool and command-line information from `.comment` sections.

Use this option with `--disassemble`.

`raw_build_attributes`

This option prints the contents of the build attributes section in raw hexadecimal form, that is, in the same form as data.

`relocation_tables`

This option prints relocation information. It has the same effect as `--text -r`.

`string_tables`

This option prints the string tables. It has the same effect as `--text -t`.

`summary` This option prints a summary of the segments and sections in a file. It is the default output of `fromelf --text`. However, the summary is suppressed by some `--info` options. Use `--emit summary` to explicitly re-enable the summary, if required.

`symbol_annotations`

This option prints symbols in disassembled code and data annotated with comments containing the respective property information.

Use this option with `--disassemble`.

`symbol_tables`

This option prints the symbol and versioning tables. It has the same effect as `--text -s`.

`vfe` This option prints information about unused virtual functions.

`whole_segments`

This option prints disassembled executables or shared libraries segment by segment even if it has a link view.

Use this option with `--disassemble`.

Multiple options can be specified in one `--emit` option followed by a comma-separated list of arguments.

See also

- *--disassemble* on page 2-24
- *--text* on page 2-55.

2.4.26 `--expandarrays`

This option prints data addresses, including arrays that are expanded both inside and outside structures.

Restrictions

This option can only be used with `--text -a`.

See also

- *--text* on page 2-55.

2.4.27 `--extract_build_attributes`

This option prints the build attributes only, either in:

- human-readable form for standard build attributes
- raw hexadecimal form for nonstandard build attributes.

Restrictions

This option can only be used in text mode.

Example

The following example shows the output for `--extract_build_attributes`:

```
========================================================================

** Object/Image Build Attributes
```

```
    'aeabi' file build attributes:
    0x000000:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000010:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000020:   41 52 4d 00                                        ARM.
        Tag_CPU_name = "ARM7TDMI"
        Tag_CPU_arch = ARM v4T (=2)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_ABI_PCS_GOT_use = Data are imported directly (=1)
        Tag_ABI_PCS_wchar_t = Size of wchar_t is 2 (=2)
        Tag_ABI_FP_denormal = This code was permitted to require that the sign of a flushed-to-zero
number be preserved in the sign of 0 (=2)
        Tag_ABI_FP_number_model = This code was permitted to use only IEEE 754 format FP numbers (=1)
        Tag_ABI_align8_needed = Code was permitted to depend on the 8-byte alignment of 8-byte data
items (=1)
        Tag_ABI_align8_preserved = Code was required to preserve 8-byte alignment of 8-byte data
objects (=1)
        Tag_ABI_enum_size = Enum values occupy the smallest container big enough to hold all values
(=1)
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion
preserved (=3)
        Tag_compatibility = 2, "ARM"

    'ARM' file build attributes:
    0x000000:   04 01 12 01                                        ....
```

See also

- *--decode_build_attributes* on page 2-20
- *--dump_build_attributes* on page 2-24
- *--text* on page 2-55.

2.4.28 `--fieldoffsets`

This option prints a list of assembly language `EQU` directives that equate C++ class or C structure field names to their offsets from the base of the class or structure. The input ELF file can be a relocatable object or an image.

Use `--output` to redirect the output to a file. Use the `INCLUDE` command from `armasm` to load the produced file and provide access to C++ classes and C structure members by name from assembly language. See the *Assembler Guide* for more information on `armasm`.

This option outputs all structure information. To output a subset of the structures, use `--select` *`select_options`*.

If you do not require a file that can be input to armasm, use the --text -a options to format the display addresses in a more readable form. The -a option only outputs address information for structures and static data in images because the addresses are not known in a relocatable object.

Restrictions

This option:

- is not available if the source file does not have debug information
- can be used only in text mode.

Example

The following examples show how to use --fieldoffsets:

- To produce an output listing to stdout that contains all the field offsets from all structures in the file inputfile.o, use:

  ```
  fromelf --fieldoffsets inputfile.o
  ```

- To produce an output file listing to outputfile.a that contains all the field offsets from structures in the file inputfile.o that have a name starting with p, use:

  ```
  fromelf --fieldoffsets --select=p* --output=outputfile.a inputfile.o
  ```

- To produce an output listing to outputfile.a that contains all the field offsets from structures in the file inputfile.o with names of tools or moretools, use:

  ```
  fromelf --fieldoffsets --select=tools.*,moretools.* --output=outputfile.a
  inputfile.o
  ```

- To produce an output file listing to outputfile.a that contains all the field offsets of structure fields whose name starts with number and are within structure field top in structure tools in the file inputfile.o, use:

  ```
  fromelf --fieldoffsets --select=tools.top.number* --output=outputfile.a
  inputfile.o
  ```

See also

- *--qualify* on page 2-46
- *--select=select_options* on page 2-50
- *--text* on page 2-55
- the *Assembler Guide*.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

2.4.29 `--fpu=list`

This option lists the supported FPU architecture names that you can use with the `--fpu=`*`name`* option.

See also

- *--fpu=name*.

2.4.30 `--fpu=`*`name`*

This option selects disassembly for a specific FPU architecture. It affects how `fromelf` interprets the instructions it finds in the input files.

Syntax

```
--fpu=name
```

Where *`name`* is the name of a supported FPU architecture.

Example

To select disassembly for the VFPv2 architecture, use:

```
--fpu=VFPv2
```

See also

- *--device=list* on page 2-21
- *--device=name* on page 2-21
- *--disassemble* on page 2-24
- *--fpu=list*
- *--info=topic[,topic,...]* on page 2-36
- *--text* on page 2-55.

2.4.31 `--globalize=`*`option`*`[,`*`option`*`,...]`

This option converts the selected symbols to global symbols.

Restrictions

You must use `--elf` with this option.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Syntax

`--globalize=`*`option`*`[,`*`option`*`,...]`

Where *option* is one of:

`object_name``::`	All symbols in ELF objects with a name matching *`object_name`* are converted to global symbols.
`object_name``::`*`symbol_name`*	All symbols in ELF objects with a name matching *`object_name`* and also a symbol name matching *`symbol_name`* are converted to global symbols.
`symbol_name`	All symbols with a symbol name matching *`symbol_name`* are converted to global symbols.

Wildcard characters ? and `*` can be used for symbolic names in *`symbol_name`* and *`object_name`* arguments. Multiple options can be specified in one `--globalize` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25
- *--hide=option[,option,...]*.

2.4.32 `--help`

This option displays a summary of the main command-line options.

This is the default if you do not specify any options or source files.

See also

- *--show_cmdline* on page 2-52
- *--version_number* on page 2-57
- *--vsn* on page 2-58.

2.4.33 `--hide=`*`option`*`[,`*`option`*`,...]`

This option changes the symbol visibility property to mark selected symbols as hidden.

Restrictions

You must use `--elf` with this option.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Syntax

`--hide=option[,option,...]`

Where *option* is one of:

`object_name``::`

All symbols in ELF objects with a name matching *`object_name`*.

`object_name``::`*`symbol_name`*

All symbols in ELF objects with a name matching *`object_name`* and also a symbol name matching *`symbol_name`*.

`symbol_name` All symbols with a symbol name matching *`symbol_name`*.

Wildcard characters `?` and `*` can be used for symbolic names in *`symbol_name`* and *`object_name`* arguments. Multiple options can be specified in one `--hide` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25
- *--show=option[,option,...]* on page 2-51.

2.4.34 `--hide_and_localize=option[,option,...]`

This option changes the symbol visibility property to mark selected symbols as hidden, and converts the selected symbols to local symbols.

Restrictions

You must use `--elf` with this option.

Syntax

`--hide_and_localize=option[,option,...]`

Where *option* is one of:

`object_name``::`

All symbols in ELF objects with a name matching *`object_name`* are marked as hidden and converted to local symbols.

`object_name::symbol_name`
: All symbols in ELF objects with a name matching `object_name` and also a symbol name matching `symbol_name` are marked as hidden and converted to local symbols.

`symbol_name`
: All symbols with a symbol name matching `symbol_name` are marked as hidden and converted to local symbols.

Wildcard characters ? and * can be used for symbolic names in `symbol_name` and `object_name` arguments. Multiple options can be specified in one `--hide_and_localize` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25.

2.4.35 `--i32`

This option produces Intel Hex-32 format output. It generates one output file for each load region in the image. You can specify the base address of the output with the `--base` option.

See also

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *Considerations when using fromelf* on page 1-3.

2.4.36 `--i32combined`

This option produces Intel Hex-32 format output. This option generates one output file for an image containing multiple load regions. You can specify the base address of the output with the `--base` option.

See also

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *Considerations when using fromelf* on page 1-3.

2.4.37 `--ignore_section=option[,option,...]`

This option specifies the sections to be ignored during a compare. Differences between the files being compared are ignored if they are in these sections.

Restrictions

You must use `--compare` with this option.

Syntax

`--ignore_section=option[,option,...]`

Where *option* is one of:

`object_name``::`	All sections in ELF objects with a name matching *`object_name`*.
`object_name``::`*`section_name`*	All sections in ELF objects with a name matching *`object_name`* and also a section name matching *`section_name`*.
`section_name`	All sections with a name matching *`section_name`*.

Wildcard characters ? and `*` can be used for symbolic names in *`section_name`* and *`object_name`* arguments. Multiple options can be specified in one `--ignore_section` option followed by a comma-separated list of arguments.

See also

- *--compare=option[,option,...]* on page 2-17
- *--ignore_symbol=option[,option,...]*
- *--relax_section=option[,option,...]* on page 2-47.

2.4.38 `--ignore_symbol=option[,option,...]`

This option specifies the symbols to be ignored during a compare. Differences between the files being compared are ignored if they are related to these symbols.

Restrictions

You must use `--compare` with this option.

Syntax

`--ignore_symbol=option[,option,...]`

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Where *option* is one of:

`object_name::`
: All symbols in ELF objects with a name matching `object_name`.

`object_name::symbol_name`
: All symbols in ELF objects with a name matching `object_name` and also a symbols name matching `symbol_name`.

`symbol_name`
: All symbols with a name matching `symbol_name`.

Wildcard characters ? and * can be used for symbolic names in `symbol_name` and `object_name` arguments. Multiple options can be specified in one `--ignore_symbol` option followed by a comma-separated list of arguments.

See also

- *--compare=option[,option,...]* on page 2-17
- *--ignore_section=option[,option,...]* on page 2-34
- *--relax_symbol=option[,option,...]* on page 2-48.

2.4.39 `--in_place`

This option enables the translation of ELF members in an input file to overwrite the previous content.

Restrictions

You must use `--elf` with this option.

Example

To remove debug information from members of a library file, enter:

```
fromelf --elf --in_place --strip=debug test.a
```

See also

- *--elf* on page 2-25
- *--strip=option[,option,...]* on page 2-53.

2.4.40 `--info=topic[,topic,...]`

This option prints information about specific topics.

Restrictions

This option can only be used in text mode.

Syntax

`--info=`*`topic`*`[,`*`topic`*`,...]`

Where *`topic`* is a comma-separated list from the following topic keywords:

`instruction_usage`
: Categorizes and lists the ARM and Thumb instructions defined in the code sections of each input file.

`function_sizes`
: Lists the names of the global functions defined in one or more input files, together with their sizes in bytes and whether they are ARM or Thumb functions.

`function_sizes_all`
: Lists the names of the local and global functions defined in one or more input files, together with their sizes in bytes and whether they are ARM or Thumb functions.

`sizes`
: Lists the `Code`, `RO Data`, `RW Data`, `ZI Data`, and `Debug` sizes for each input object and library member in the image. Using this option implies `--info=sizes,totals`.

`totals`
: Lists the totals of the `Code`, `RO Data`, `RW Data`, `ZI Data`, and `Debug` sizes for input objects and libraries.

The output from `--info=sizes,totals` always includes the padding values in the totals for input objects and libraries.

Note

Spaces are not permitted between topic keywords in the list. For example, you can enter `--info=sizes,totals` but not `--info=sizes, totals`.

See also

- *--text* on page 2-55.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

2.4.41 *input_file*

This option specifies the ELF file or library file to be processed. Multiple input files are supported if you:

- output `--text` format
- use the `--compare` option
- use `--elf` with `--in_place`
- specify an output directory using `--output`.

Usage

If *`input_file`* is a scatter-loaded image that contains more than one load region and the output format is one of `--bin`, `--m32`, `--i32`, or `--vhx`, `fromelf` creates a separate file for each load region.

If *`input_file`* is a scatter-loaded image that contains more than one load region and the output format is either `--m32combined` or `--i32combined`, `fromelf` creates a single file containing all load regions.

If *`input_file`* is an archive, you can process all files, or a subset of files, in that archive. To process a subset of files in the archive, specify a filter after the archive name as follows:

```
archive.a(filter_pattern)
```

where *`filter_pattern`* specifies a member file. To specify a subset of files use the following wildcard characters:

*	to match zero or more characters
?	to match any single character.

Note

On Unix systems your shell typically requires the parentheses and these characters to be escaped with backslashes. Alternatively, enclose the archive name and filter in single quotes, for example:

```
'archive.a(??str*)'
```

Any files in the archive that are not processed are included in the output archive together with the processed files.

Example

The following example strips debug information from all files in the archive beginning with s, and creates a new archive, `my_archive.a`, containing the processed and unprocessed files.

```
fromelf --elf --strip=debug archive.a(s*.o) --output=my_archive.a
```

See also

- *Processing ELF files in an archive* on page 2-5
- *--bin* on page 2-12
- *--compare=option[,option,...]* on page 2-17
- *--elf* on page 2-25
- *--i32* on page 2-34
- *--i32combined* on page 2-34
- *--in_place* on page 2-36
- *--m32* on page 2-42
- *--m32combined* on page 2-43
- *--output=destination* on page 2-44
- *--text* on page 2-55
- *--vhx* on page 2-57.

2.4.42 `--interleave=`*option*

This option inserts the original source code as comments into the disassembly if debug information is present.

Use this option with `--emit=code`, `--text -c` or `--disassemble`.

Syntax

`--interleave=`*option*

Where *option* can be one of the following:

`line_directives`

interleaves `#line` directives containing filenames and line numbers of the disassembled instructions.

`line_numbers` interleaves comments containing filenames and line numbers of the disassembled instructions.

`none`	interleaving is disabled. This is useful if you have a generated makefile where the fromelf command has multiple options in addition to `--interleave`. You can then specify `--interleave=none` as the last option to ensure that interleaving is disabled without having to reproduce the complete fromelf command.
`source`	interleaves comments containing source code. If the source code is no longer available then `fromelf` interleaves in the same way as `line_numbers`.
`source_only`	interleaves comments containing source code. If the source code is no longer available then `fromelf` does not interleave that code.

Default

The default is `--interleave=none`.

See also

- *--disassemble* on page 2-24
- *--emit=option[,option,...]* on page 2-25
- *--source_directory=path* on page 2-53
- *--text* on page 2-55.

2.4.43 `--licretry`

If you are using floating licenses, this option makes up to 10 attempts to obtain a license when you invoke `fromelf`.

Usage

Use this option if your builds are failing to obtain a license from your license server, and only after you have ruled out any other problems with the network or the license server setup.

It is recommended that you place this option in the `RVCT40_FROMELFOPT` environment variable. In this way, you do not have to modify your build files.

See also

- *--licretry* on page 2-82 in the *Compiler Reference Guide*
- *--licretry* on page 2-56 in the *Linker Reference Guide*
- *Command syntax* on page 3-2 in the *Assembler Guide*

- *Environment variables used by RVCT* on page 1-7 in the *RealView Compilation Tools Essentials Guide*
- *FLEXnet for ARM Tools License Management Guide*.

2.4.44 `--linkview`, `--no_linkview`

This option controls the section-level view from the ELF image.

`--no_linkview` discards the section-level view and retains only the segment-level view (load time view). Discarding the link-view section level eliminates:

- the section header table
- the section header string table
- the string table
- the symbol table
- all debug sections.

All that is left in the output is the program header table and the program segments. According to the *System V Application Binary Interface* specification, these are all that a program loader can rely on being present in an ELF file.

Restrictions

The following restrictions apply:

- you must use `--elf` with `--linkview` and `--no_linkview`
- do not use the `--no_linkview` option with SysV images.

Example

To get ELF format output, enter:

```
fromelf --no_linkview --elf image.axf --output=image_nlk.axf
```

See also

- *--elf* on page 2-25
- *--strip=option[,option,...]* on page 2-53
- *System V Application Binary Interface – DRAFT – 17 December 2003* specification.

2.4.45 `--localize=`*`option`*`[,`*`option,...`*`]`

This option converts the selected symbols to local symbols.

Restrictions

You must use `--elf` with this option .

Syntax

`--localize=`*`option`*`[,`*`option`*`,...]`

Where *option* is one of:

`object_name``::`

All symbols in ELF objects with a name matching *`object_name`* are converted to local symbols.

`object_name``::`*`symbol_name`*

All symbols in ELF objects with a name matching *`object_name`* and also a symbol name matching *`symbol_name`* are converted to local symbols.

`symbol_name` All symbols with a symbol name matching *`symbol_name`* are converted to local symbols.

You can:

- use wildcard characters ? and `*` for symbolic names in *`symbol_name`* and *`object_name`* arguments
- specify multiple options in one `--localize` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25
- *--hide=option[,option,...]* on page 2-32.

2.4.46 `--m32`

This option produces Motorola 32-bit format (32-bit S-records) output. It generates one output file for each load region in the image. You can specify the base address of the output with the `--base` option.

See also

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *Considerations when using fromelf* on page 1-3.

2.4.47 `--m32combined`

This option produces Motorola 32-bit format (32-bit S-records) output. This option generates one output file for an image containing multiple load regions. You can specify the base address of the output with the `--base` option.

See also

- *--base [[object_file::]load_region_ID=]num* on page 2-11
- *Considerations when using fromelf* on page 1-3.

2.4.48 `--only=`*`section_name`*

This option forces the output to display only the named section.

Syntax

`--only=`*`section_name`*

Where *`section_name`* is the name of the section to be displayed.

You can:

- use wildcard characters ? and `*` for a section name
- use multiple `--only` options to specify additional section to display.

Example

The following examples show how to use `--only`:

- To display only the symbol table, `.symtab`:

  ```
  fromelf --only=.symtab --text -s test.axf
  ```

- To display all `ER`*`n`* sections:

  ```
  fromelf --only=ER? test.axf
  ```

- To display the `HEAP` section and all symbol and string table sections:

  ```
  fromelf --only=HEAP --only=.*tab --text -s -t test.axf
  ```

See also

- *--text* on page 2-55.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

2.4.49 `--output=`*`destination`*

This option specifies the name of the output file, or the name of the output directory if multiple output files are created.

Syntax

`--output=`*destination*

Where *destination* can be either a file or a directory. For example:

`--output=foo`	is the name of an output file
`--output=foo/`	is the name of an output directory.

Usage

Usage with `--bin` or `--elf`:

- You can specify a single input file and a single output filename.
- If you specify many input files and use `--elf`, you can use `--in_place` to write the output of processing each file over the top of the input file.
- If you specify many input filenames and specify an output directory, then the output from processing each file is written into the output directory. Each output filename is derived from the corresponding input file. Therefore, specifying an output directory in this way is the only method of converting many ELF files to a binary or hexadecimal format in a single run of `fromelf`.
- If you specify an archive file as the input, then the output file is also an archive. For example, the following command creates an archive file called `output.o`:

  ```
  fromelf --elf --strip=debug mylib.a --output=output.o
  ```

- If you specify a pattern in parentheses to select a subset of objects from an archive, fromelf only converts the subset. All the other objects are passed through to the output archive, unchanged.

See also

- *--bin* on page 2-12
- *--elf* on page 2-25
- *--text* on page 2-55.

2.4.50 `--privacy`

The effect of this option is different for images and object files.

For images, this option:

- changes section names to a default value, for example, changes code section names to `.text`
- removes the complete symbol table in the same way as `--strip symbols`
- removes the `.comment` section name, and is marked as `[Anonymous Section]` in the `fromelf --text` output.

For object files, this option:

- Changes section names to a default value, for example, changes code section names to `.text`.
- Keeps mapping symbols and build attributes in the symbol table.
- Removes those local symbols that can be removed without loss of functionality.

  Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.

Use this option to help protect your code in images and objects that are delivered to third parties.

See also

- *Protecting code in images and objects with fromelf* on page 2-2
- *--strip=option[,option,...]* on page 2-53
- *--locals, --no_locals* on page 2-58 in the *Linker Reference Guide*.
- *--privacy* on page 2-68 in the *Linker Reference Guide*.

2.4.51 `--project=`*`filename`*, `--no_project=`*`filename`*

Controls the loading of the specified project template file.

Syntax

`--project=`*`filename`*

Where *`filename`* is the name of a project template file.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Note

To use `filename` as a default project file, set the `RVDS_PROJECT` environment variable to `filename`.

`--no_project` prevents the default project template file specified by the environment variable `RVDS_PROJECT` from being used.

Restrictions

Options from a project template file are only set when they do not conflict with options already set on the command-line. If an option from a project template file conflicts with an existing command-line option, the command-line option takes precedence.

Example

Consider the following project template file:

```
<!-- suiteconf.cfg -->
<suiteconf name="Platform Baseboard for ARM926EJ-S">
    <tool name="fromelf">
        <cmdline>
            --cpu=ARM926EJ-S
            --fpu=vfpv2
        </cmdline>
    </tool>
</suiteconf>
```

When the `RVDS_PROJECT` environment variable is set to point to this file, the command:

```
fromelf foo.o
```

results in an actual command line of:

```
fromelf --cpu=ARM926EJ-S --fpu=vfpv2 foo.o
```

See also

- *--reinitialize_workdir* on page 2-47
- *--workdir=directory* on page 2-61.

2.4.52 `--qualify`

This option modifies the effect of the `--fieldoffsets` option so that the name of each output symbol includes an indication of the source file containing the relevant structure. This enables the `--fieldoffsets` option to produce functional output even if two source files define different structures with the same name.

Example

A structure called `foo` is defined in two headers for example, `one.h` and `two.h`.

Using `fromelf --fieldoffsets`, the linker might define the following symbols:

- `foo.a`, `foo.b`, and `foo.c`
- `foo.x`, `foo.y`, and `foo.z`

Using `fromelf --qualify --fieldoffsets`, the linker defines the following symbols:

- `oneh_foo.a`, `oneh_foo.b` and `oneh_foo.c`
- `twoh_foo.x`, `twoh_foo.y` and `twoh_foo.z`

See also

- *--fieldoffsets* on page 2-29.

2.4.53 `--reinitialize_workdir`

This option enables you to reinitialize the project template working directory set using `--workdir`.

When the directory set using `--workdir` refers to an existing working directory containing modified project template files, specifying this option causes the working directory to be deleted and recreated with new copies of the original project template files.

Restrictions

You must use `--workdir` with this option.

See also

- *--project=filename, --no_project=filename* on page 2-45
- *--workdir=directory* on page 2-61.

2.4.54 `--relax_section=`*`option`*`[,`*`option`*`,...]`

This option changes the severity of a compare report for the specified sections to warnings rather than errors.

Restrictions

You must use `--compare` with this option.

Syntax

`--relax_section=`*`option`*`[,`*`option`*`,...]`

Where *option* is one of:

`object_name``::`

All sections in ELF objects with a name matching *`object_name`*.

`object_name``::`*`section_name`*

All sections in ELF objects with a name matching *`object_name`* and also a section name matching *`section_name`*.

`section_name` All sections with a name matching *`section_name`*.

You can:

- use wildcard characters ? and * for symbolic names in *`section_name`* and *`object_name`* arguments
- specify multiple options in one `--relax_section` option followed by a comma-separated list of arguments.

See also

- *--compare=option[,option,...]* on page 2-17
- *--ignore_section=option[,option,...]* on page 2-34
- *--relax_symbol=option[,option,...]*.

2.4.55 `--relax_symbol=`*`option`*`[,`*`option`*`,...]`

This option changes the severity of a compare report for the specified symbols to warnings rather than errors.

Restrictions

You must use `--compare` with this option.

Syntax

`--relax_symbol=`*`option`*`[,`*`option`*`,...]`

Where *option* is one of:

`object_name``::`

All symbols in ELF objects with a name matching *`object_name`*.

object_name::*section_name*

All symbols in ELF objects with a name matching *object_name* and also a symbol name matching *symbol_name*.

symbol_name All symbols with a name matching *symbol_name*.

You can:

- use wildcard characters ? and * for symbolic names in *symbol_name* and *object_name* arguments
- specify multiple options in one `--relax_symbol` option followed by a comma-separated list of arguments.

See also

- *--compare=option[,option,...]* on page 2-17
- *--ignore_symbol=option[,option,...]* on page 2-35
- *--relax_section=option[,option,...]* on page 2-47.

2.4.56 `--rename=`*option*`[`*,option,...*`]`

This option renames the specified symbol in an output ELF object.

Restrictions

You must use `--elf` and `--output` with this option.

Syntax

`--rename=`*option*`[,`*option*`,...]`

Where *option* is one of:

object_name::*old_symbol_name*=*new_symbol_name*

This replaces all symbols in the ELF object *object_name* that have a symbol name matching *old_symbol_name*.

old_symbol_name=*new_symbol_name*

This replaces all symbols that have a symbol name matching *old_symbol_name*.

You can:

- use wildcard characters ? and * for symbolic names in *old_symbol_name*, *new_symbol_name* and *object_name* arguments

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

- specify multiple options in one `--rename` option followed by a comma-separated list of arguments.

Example

This example renames the `clock` symbol in the `dhrystone.axf` image to `myclock`, and creates a new file called `dhry.axf`:

```
fromelf --elf --rename=clock=myclock --output=dhry.axf dhrystone.axf
```

See also

- *--elf* on page 2-25
- *--output=destination* on page 2-44.

2.4.57 `--select=`*`select_options`*

This option selects only those fields that match a specified pattern list.

Use this option with either `--fieldoffsets` or `--text -a`.

Syntax

`--select=`*select_options*

Where *select_options* is a list of patterns to match. Use special characters to select multiple fields:

- Use a comma-separated list to join options together, for example:

  `a*,b*,c*`

- Use the wildcard character `*` to match any name.
- Use the wildcard character ? to match any single letter.
- Prefix the *`select_options`* string with `+` to specify the fields to include. This is the default behavior.
- Prefix the *`select_options`* string with `~` to specify the fields to exclude.

If you are using a special character on Unix platforms, you must enclose the options in quotes to prevent the shell expanding the selection.

See also

- *--fieldoffsets* on page 2-29
- *--text* on page 2-55.

2.4.58 `--show=option[,option,...]`

This option changes the symbol visibility property of the selected symbols, to mark them with default visibility.

Restrictions

You must use `--elf` with this option.

Syntax

`--show=option[,option,...]`

Where *option* is one of:

`object_name::`
: All symbols in ELF objects with a name matching `object_name` are marked as having default visibility.

`object_name::symbol_name`
: All symbols in ELF objects with a name matching `object_name` and also a symbol name matching `symbol_name` are marked as having default visibility.

`symbol_name`
: All symbols with a symbol name matching `symbol_name` are marked as having default visibility.

You can:

- use wildcard characters ? and * for symbolic names in `symbol_name` and `object_name` arguments
- specify multiple options in one `--show` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25
- *--hide=option[,option,...]* on page 2-32.

2.4.59 `--show_and_globalize=option[,option,...]`

This option changes the symbol visibility property of the selected symbols, to mark them with default visibility, and converts the selected symbols to global symbols.

*Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,*

Restrictions

You must use `--elf` with this option.

Syntax

`--show_and_globalize=option[,option,...]`

Where *option* is one of:

`object_name::`	All symbols in ELF objects with a name matching `object_name`.
`object_name::symbol_name`	All symbols in ELF objects with a name matching `object_name` and also a symbol name matching `symbol_name`.
`symbol_name`	All symbols with a symbol name matching `symbol_name`.

You can:

- use wildcard characters `?` and `*` for symbolic names in `symbol_name` and `object_name` arguments
- specify multiple options in one `--show_and_globalize` option followed by a comma-separated list of arguments.

See also

- *--elf* on page 2-25.

2.4.60 `--show_cmdline`

This option shows how `fromelf` has processed the command line. It shows the command-line after processing by `fromelf`, and can be useful to check:

- the command-line a build system is using
- how `fromelf` is interpreting the supplied command-line, for example, the ordering of command line options.

The commands are shown in their preferred form, and the contents of any via files are expanded.

2.4.61 `--source_directory=path`

This option explicitly specifies the directory of the source code. By default, the source code is assumed to be located in a directory relative to the ELF input file. You can use this option multiple times to specify a search path involving multiple directories.

Use this option with `--interleave`.

See also

- *--interleave=option* on page 2-39.

2.4.62 `--strip=option[,option,...]`

This option protects IP in images and objects that are delivered to third parties. It can also be used to help reduce the size of the output image.

Restrictions

You must use `--elf` with this option.

Syntax

`--strip=option[,option,...]`

Where *option* is one of:

`all` For object modules, this option removes all debug, comments, notes and symbols from the ELF file. For executables, this options works the same as `--no_linkview`.

——— **Note** ———

Do not use the `--strip=all` option with SysV images.

———————————

`debug` Removes all debug sections from the ELF file.

`comment` Removes the **`.comment`** section from the ELF file.

`filesymbols` The `STT_FILE` symbols are removed from the ELF file.

`localsymbols` The effect of this option is different for images and object files.

For images, this option removes all local symbols, including mapping symbols, from the output symbol table.

For object files, this option:

- Keeps mapping symbols and build attributes in the symbol table.

- Removes those local symbols that can be removed without loss of functionality.

  Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.

`notes` Removes the **`.notes`** section from the ELF file.

`pathnames` Removes the path information from all symbols with type `STT_FILE`. For example, an `STT_FILE` symbol with the name `C:\work\myobject.o` is renamed to `myobject.o`.

`symbols` The effect of this option is different for images and object files.

For images, this option removes the complete symbol table, and all static symbols. If any of these static symbols are used as a static relocation target, then these relocations are also removed. In all cases, `STT_FILE` symbols are removed.

For object files, this option:

- Keeps mapping symbols and build attributes in the symbol table.
- Removes those local symbols that can be removed without loss of functionality.

  Symbols that cannot be removed, such as the targets for relocations, are kept. For these symbols, the names are removed. These are marked as `[Anonymous Symbol]` in the `fromelf --text` output.

Note

Stripping the symbols, pathnames or file symbols might make the file harder to debug.

Example

To produce a new output file without debug from an ELF file originally produced with debug, use:

```
fromelf --strip=debug,symbols --elf --output=outfile.axf infile.axf
```

See also

- *--elf* on page 2-25
- *--linkview, --no_linkview* on page 2-41
- *--privacy* on page 2-45

- *About mapping symbols* on page 4-2 in the *Linker User Guide*
- *--locals, --no_locals* on page 2-58 in the *Linker Reference Guide*
- *--privacy* on page 2-68 in the *Linker Reference Guide*.

2.4.63 `--symbolversions`, `--no_symbolversions`

This option turns off the decoding of symbol version tables. See *Symbol versioning* on page 4-19 and the *Base Platform ABI for the ARM Architecture* (BPABI) for more information.

2.4.64 `--text`

This option prints image information in text format. You can decode an ELF image or ELF object file using this option.

If you do not specify a code output format, `--text` is assumed. That is, you can specify one or more options without having to specify `--text`. For example, `fromelf -a` is the same as `fromelf --text -a`.

If you specify a code output format, such as `--bin`, then any `--text` options are ignored.

If *`destination`* is not specified with the `--output` option, or `--output` is not specified, the information is displayed on `stdout`.

Syntax

`--text [`*`options`*`]`

Where *`options`* specifies what is displayed, and can be one or more of the following:

`-a` Prints the global and static data addresses (including addresses for structure and union contents).

This option can only be used on files containing debug information. If no debug information is present, a warning is displayed.

Use the `--select` option to output a subset of the data addresses.

If you want to view the data addresses of arrays, expanded both inside and outside structures, use the `--expandarrays` option with this text category.

`-c` This option disassembles code, alongside a dump of the original binary data being disassembled and the addresses of the instructions.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Note

Unlike `--disassemble`, the disassembly cannot be input to the ARM assembler.

`-d`	Prints contents of the data sections.
`-e`	Decodes exception table information for objects. Use with `-c` when disassembling images.
`-g`	Prints debug information.
`-r`	Prints relocation information.
`-s`	Prints the symbol and versioning tables.
`-t`	Prints the string tables.
`-v`	Prints detailed information on each segment and section header of the image.
`-w`	Eliminates line wrapping.
`-y`	Prints dynamic segment contents.
`-z`	Prints the code and data sizes.

These options are only recognized in text mode.

Example

The following examples show how to use `--text`:

- To produce a plain text output file that contains the disassembled version of an ELF image and the symbol table, use:

  ```
  fromelf --text -c -s --output=outfile.lst infile.axf
  ```

- To list to `stdout` all the global and static data variables and all the structure field addresses, use:

  ```
  fromelf --text -a --select=* infile.axf
  ```

- To produce a text file containing all of the structure addresses in `inputfile.axf` but none of the global or static data variable information, use:

  ```
  fromelf --text -a --select=*.* --output=structaddress.txt infile.axf
  ```

- To produce a text file containing addresses of the nested structures only, use:

  ```
  fromelf --text -a --select=*.*.* --output=structaddress.txt infile.axf
  ```

- To produce a text file containing all of the global or static data variable information in `inputfile.axf` but none of the structure addresses, use:

  ```
  fromelf --text -a --select=*,~*.* --output=structaddress.txt infile.axf
  ```

See also

- *--cpu=name* on page 2-19
- *--disassemble* on page 2-24
- *--emit=option[,option,...]* on page 2-25
- *--expandarrays* on page 2-28
- *--info=topic[,topic,...]* on page 2-36
- *--interleave=option* on page 2-39
- *--output=destination* on page 2-44
- *--select=select_options* on page 2-50
- *Getting information about images* on page 3-41.

2.4.65 --version_number

This option displays the version of `fromelf` you are using.

Syntax

```
fromelf --version_number
```

`fromelf` displays the version number in the format `nnnbbb`, where:

- `nnn` is the version number
- `bbb` is the build number.

Example

Version 4.0.0 build 821 is displayed as `400821`.

See also

- *--help* on page 2-32
- *--vsn* on page 2-58

2.4.66 `--vhx`

This option produces Byte oriented (Verilog Memory Model) hexadecimal format output. This format is suitable for loading into the memory models of *Hardware Description Language* (HDL) simulators. You can split output from this option into multiple files with the `--widthxbanks` option.

See also

- *--widthxbanks* on page 2-59
- *Considerations when using fromelf* on page 1-3.

2.4.67 `--vsn`

This option displays `fromelf` version information, including the type of license being used. For example:

```
>fromelf --vsn
ARM FromELF, RVCT4.0 [Build 821]
license_type
Software supplied by: ARM Limited
```

See also

- *--help* on page 2-32
- *--version_number* on page 2-57.

2.4.68 `-w`

This option causes some text output information that usually appears on multiple lines to be displayed on a single line.

This makes the output easier to parse with text processing utilities such as Perl.

For example:

```
> fromelf --text -w -c test.axf
========================================================================

** ELF Header Information
.
.
.
========================================================================

** Section #1 'ER_RO' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]    Size   : 4516 bytes (alignment 4)
Address: 0x00008000    $a
.
.
.
** Section #2 'ER_RW' (SHT_PROGBITS) [SHF_ALLOC + SHF_WRITE]    Size   : 16 bytes (alignment 4)
Address: 0x000091a4
** Section #3 'ER_ZI' (SHT_NOBITS) [SHF_ALLOC + SHF_WRITE]    Size   : 336 bytes (alignment 4)
Address: 0x000091b4
** Section #4 '.debug_frame' (SHT_PROGBITS)    Size   : 2104 bytes
** Section #5 '.symtab' (SHT_SYMTAB)    Size   : 7088 bytes (alignment 4)    String table #6 '.strtab'
```

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

```
Last local symbol no. 293
** Section #6 '.strtab' (SHT_STRTAB)    Size   : 4696 bytes
** Section #7 '.note' (SHT_NOTE)    Size   : 28 bytes (alignment 4)
** Section #8 '.comment' (SHT_PROGBITS)    Size   : 1140 bytes
** Section #9 '.shstrtab' (SHT_STRTAB)    Size   : 76 bytes
```

See also

- *--text* on page 2-55.

2.4.69 `--`*`width`*`x`*`banks`*

This option outputs multiple files for multiple memory banks.

`fromelf` uses the last specified configuration if more than one configuration is specified.

Restrictions

You must use `--output` with this option.

Syntax

`--`*`width`*`x`*`banks`*

Where:

`banks`	specifies the number of memory banks in the target memory system. It determines the number of output files that are generated for each load region.
`width`	is the width of memory in the target memory system (8-bit, 16-bit, 32-bit, or 64-bit).

Valid configurations are:

```
--8x1
--8x2
--8x4
--16x1
--16x2
--32x1
--32x2
--64x1
```

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

If the image has one load region, `fromelf` generates the same number of files as the number of *`banks`* specified. The filenames are derived from the `--output=`*`destination`* argument, using the following naming conventions:

- If there is one memory bank (*`banks`*=1) the output file is named *`destination`*.
- If there are multiple memory banks (*`banks`*>1), `fromelf` generates *`banks`* number of files named *`destinationN`* where *`N`* is in the range `0` to *`banks`*`-1`. If you specify a file extension for the output filename, then the number *`N`* is placed before the file extension. For example:

  ```
  fromelf --vhx --8x2 test.axf --output=test.txt
  ```

  This generates two files named `test0.txt` and `test1.txt`.

If the image has multiple load regions, `fromelf` creates a directory named *`destination`* and generates *`banks`* files for each load region in that directory. The files for each load region are named *`load_regionN`* where *load_region* is the name of the load region, and *`N`* is in the range `0` to *`banks`*`-1`. For example:

```
fromelf --vhx --8x2 multiload.axf --output=regions
```

This might produce the following files in the `regions` directory:

```
EXEC_ROM0
EXEC_ROM1
RAM0
RAM1
```

The memory width specified by *`width`* controls the amount of memory that is stored in a single line of each output file. The size of each output file is the size of memory to be read divided by the number of files created. For example:

- `fromelf --vhx --8x4 test.axf --output=file` produces four files (`file0`, `file1`, `file2`, and `file3`). Each file contains lines of single bytes, for exmaple:

  ```
  00
  00
  2D
  00
  2C
  8F
  ...
  ```

- `fromelf --vhx --16x2 test.axf --output=file` produces two files (`file0` and `file1`). Each file contains lines of two bytes, for example:

  ```
  0000
  002D
  002C
  ...
  ```

See also

- *--output=destination* on page 2-44
- *--vhx* on page 2-57.

2.4.70 `--workdir=`*`directory`*

This option enables you to provide a working directory for a project template.

Note

Project templates only require working directories if they include files, for example, RealView Debugger configuration files.

Syntax

`--workdir=`*`directory`*

Where *`directory`* is the name of the project directory.

Restrictions

If you specify a project working directory using `--workdir`, then you must specify a project file using `--project`.

Errors

An error message is produced if you try to use `--project` without `--workdir` and `--workdir` is required.

See also

- *--project=filename, --no_project=filename* on page 2-45
- *--reinitialize_workdir* on page 2-47.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

Chapter 3
Using armar

This chapter describes the ARM® librarian, `armar`, provided with ARM RealView® Compilation Tools. It includes:

- *Using command-line options* on page 3-2
- *Command-line options* on page 3-5.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

3.1 Using command-line options

When specifying command-line options, the following rules apply depending on the type of option:

Single-letter options

All single-letter options are preceded by a single dash `-`. If the option has an argument, you can use a space between the option and the argument, or the argument can immediately follow the option.

For example:

```
-a obj2.o
```

```
-aobj2.o
```

Keyword options

All keyword options are preceded by a double dash `--`. An = or space character is required between the option and the argument.

For example:

```
--diag_style=ide
```

```
--diag_style ide
```

3.1.1 Invoking armar

The `armar` command-line syntax is:

```
armar  [diagnostic-options] [help-options] [input-options] [managing-options]
[order-options] [output-options] [project-template-options] [updating-options]
archive [file_list]
```

`diagnostic-options`

Use the following options to control diagnostic messages:

- *-c* on page 3-6
- *--diag_style={arm|ide|gnu}* on page 3-7
- *--show_cmdline* on page 3-12.

`help-options`

Use the following options to shows the main command-line options and the version number of the librarian:

- *--help* on page 3-8
- *--vsn* on page 3-14.

`input-options`

Use the following option to take command-line arguments from a file:

- *--via=file* on page 3-14.

`managing-options`

Use the following options to provide library entry points, filenames, and directory information:

- *--entries* on page 3-7
- *file_list* on page 3-8
- *archive* on page 3-5.

`order-options`

Use the following options to control the placing of library files:

- *-a pos_name* on page 3-5
- *-b pos_name* on page 3-6
- *-i pos_name* on page 3-8
- *-m pos_name* on page 3-9

`output-options`

Use the following options to control librarian output:

- *-n* on page 3-9
- *-p* on page 3-9
- *-s* on page 3-12
- *--sizes* on page 3-12
- *-t* on page 3-13
- *-v* on page 3-13
- *-x* on page 3-15
- *--zs* on page 3-15
- *--zt* on page 3-16.

`project-template-options`

Use the following options to control the use of project templates:

- *--project=filename, --no_project=filename* on page 3-10
- *--reinitialize_workdir* on page 3-11
- *--workdir=directory* on page 3-14.

`updating-options`

Use the following options to create, update, and delete library files:

- *-C* on page 3-6

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

- *--create* on page 3-6
- *-d* on page 3-6
- *-r* on page 3-11
- *-T* on page 3-13
- *-u* on page 3-13.

`archive`	The filename of the library, and is described in *archive* on page 3-5. A library file must always be specified.
`file_list`	The list of files to be processed, and is described in *file_list* on page 3-8.

3.1.2 Ordering command-line options

In general, command-line options can appear in any order. However, the effects of some options depend on how they are combined with other related options.

Where options override other options on the same command line, the options that appear closer to the end of the command-line take precedence. Where an option does not follow this rule, this is noted in the description for that option. Use the `--show_cmdline` option to see how `armar` has processed the command line. The commands are shown normalized, and the contents of any via files are expanded.

See also

- *--show_cmdline* on page 3-12
- *--via=file* on page 3-14.

3.2 Command-line options

This section lists the command-line options supported by the librarian in alphabetical order.

Note

In RealView Compilation Tools v3.0 and later, `armar` command-line options must be preceded by a `-`. This is different from earlier versions of `armar`, and from some third-party archivers.

Note

You can use special characters to select multiple symbolic names in some `armar` arguments:

- The wildcard character * can be used to match any name.
- The wildcard character ? can be used to match any single character.

If you are using a special character on Unix platforms, you must enclose the options in quotes to prevent the shell expanding the selection.

For example, enter `'*,~*.*'` instead of `*,~*.*`.

3.2.1 *archive*

Specifies the location of the library to be created, modified, or read.

Note

If you include a list of files in *`file_list`*, they must be specified after the library file.

3.2.2 `-a` *pos_name*

Places new files in the library after the file *`pos_name`*. The effect of this option is negated if you include `-b` (or `-i`) on the same command line.

Example

To add or replace files immediately after `obj2.o` in `mylib`, use:

```
armar -r -a obj2.o mylib obj3.o obj4.o ...
```

See also

- *-b pos_name* on page 3-6

- *-i pos_name* on page 3-8
- *-m pos_name* on page 3-9
- *-r* on page 3-11.

3.2.3 `-b` *`pos_name`*

Places new files in the library before the file *`pos_name`*. This option takes precedence if you include `-a` on the same command line.

See also

- *-a pos_name* on page 3-5
- *-i pos_name* on page 3-8
- *-m pos_name* on page 3-9
- *-r* on page 3-11.

3.2.4 `-c`

Suppresses the diagnostic message normally written to `stderr` when a library is created.

3.2.5 `-C`

Instructs the librarian not to replace existing files with like-named files when performing extractions. This option is useful when `-T` is also used to prevent truncated filenames from replacing files with the same prefix.

See also

- *-T* on page 3-13.

3.2.6 `--create`

Creates a new library containing only the files specified in *`file_list`*. If the library already exists, its previous contents are discarded.

Example

To create a new library by adding all object files in the current directory, use:

```
armar --create mylib *.o
```

3.2.7 `-d`

Deletes one or more files specified in *`file_list`* from the library.

3.2.8 `--diag_style={arm|ide|gnu}`

Specifies the style used to display diagnostic messages.

Syntax

`--diag_style=`*`string`*

Where *`string`* is one of:

`arm`	Display messages using the ARM style.
`ide`	Include the line number and character count for any line that is in error. These values are displayed in parentheses.
`gnu`	Display messages in the format used by GNU.

Default

If you do not specify a `--diag_style` option, it assumes `--diag_style=arm`.

3.2.9 `--entries`

Lists all object files in the library that have an entry point defined using the assembler `ENTRY` directive.

The format for the listing is:

`ENTRY at offset` *`num`* `in section` *`name`* `of` *`member`*

Example

The following example lists the entry point of each object file in `myasm.a`:

```
> armar --entries myasm.a
ENTRY at offset 0 in section adrlabel of adrlabel.o
ENTRY at offset 0 in section ARMex of armex.o
ENTRY at offset 0 in section Block of blocks.o
ENTRY at offset 0 in section Jump of jump.o
ENTRY at offset 0 in section LDRlabel of ldrlabel.o
ENTRY at offset 0 in section Loadcon of loadcon.o
ENTRY at offset 0 in section StrCopy of strcopy.o
ENTRY at offset 0 in section subrout of subrout.o
ENTRY at offset 0 in section Tblock of tblock.o
ENTRY at offset 0 in section ThumbSub of thumbsub.o
ENTRY at offset 0 in section Word of word.o
```

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

See also

- *--sizes* on page 3-12
- *--zt* on page 3-16
- *ENTRY* on page 7-76 in the *Assembler Guide*:

3.2.10 *file_list*

A list of files to process. Each file is fully specified by its path and name. The path can be absolute, relative to drive and root, or relative to the current directory.

Note

The list of files must be specified after the library file.

Only the filename at the end of the path is used when comparing against the names of files in the library. If two or more path operands end with the same filename, the results are unspecified. You can use the wild characters `*` and ? to specify files.

If one of the files is a library, `armar` copies all members from the input library to the destination library. The order of members on the command line is preserved. Therefore, supplying a library file is logically equivalent to supplying all of its members in the order that they are stored in the library.

3.2.11 `--help`

Displays a summary of the main command-line options.

This is the default if you do not specify any options or source files.

See also

- *--vsn* on page 3-14.

3.2.12 `-i` *pos_name*

Places new files in the library before the member *pos_name* (equivalent to `-b` *pos_name*).

See also

- *-a pos_name* on page 3-5
- *-b pos_name* on page 3-6
- *-m pos_name* on page 3-9
- *-r* on page 3-11.

3.2.13 `-m` *`pos_name`*

Moves files. If `-a`, `-b`, or `-i` with *`pos_name`* is specified, files are moved to the new position. Otherwise, move files to the end of library.

See also

- *-a pos_name* on page 3-5
- *-b pos_name* on page 3-6
- *-i pos_name* on page 3-8.

3.2.14 `-n`

Suppresses the creation of a symbol table in the library.

Note

By default, `armar` always creates a symbol table when you create a library of object files.

You can recreate the symbol table in the library using the `-s` option.

Example

To create a library without a symbol table, use:

```
armar -n --create mylib.a *.obj
```

See also

- *-s* on page 3-12.

3.2.15 `--new_files_only`

Updates older files. When used with the `-r` option, files in the library are replaced only if the corresponding file has a modification time that is newer than the modification time of the file in the library.

See also

- *-r* on page 3-11
- *-u* on page 3-13.

3.2.16 `-p`

Prints the contents of files in *`library`* to `stdout`.

3.2.17 `--project=`*`filename`*, `--no_project=`*`filename`*

Instructs the compiler to load the specified project template file.

Syntax

`--project=`*`filename`*

Where *`filename`* is the name of a project template file.

Note

To use *`filename`* as a default project file, set the `RVDS_PROJECT` environment variable to *`filename`*.

`--no_project` prevents the default project template file specified by the environment variable `RVDS_PROJECT` from being used.

Restrictions

Options from a project template file are only set when they do not conflict with options already set on the command line. If an option from a project template file conflicts with an existing command-line option, the command-line option takes precedence.

Example

Consider the following project template file:

```
<!-- suiteconf.cfg -->
<suiteconf name="Platform Baseboard for ARM926EJ-S">
    <tool name="armar">
        <cmdline>
            --cpu=ARM926EJ-S
            --fpu=vfpv2
        </cmdline>
    </tool>
</suiteconf>
```

When the `RVDS_PROJECT` environment variable is set to point to this file, the command:

```
armar mylib foo.o
```

results in an actual command line of:

```
armar --cpu=ARM926EJ-S --fpu=vfpv2 mylib foo.o
```

See also

- *--reinitialize_workdir*
- *--workdir=directory* on page 3-14.

3.2.18 `-r`

Replaces, or adds, files in the library. If the library does not exist, a new library file is created and a diagnostic message is written to standard error. `-q` is an alias for `-r`.

If *`file_list`* is not specified and the library exists, the results are undefined. Files that replace existing files do not change the order of the library.

If the `-u` option is used, then only those files with dates of modification later than the library files are replaced.

If the `-a`, `-b`, or `-i` option is used, then *`pos_name`* must be present and specifies that new files are to be placed after (`-a`) or before (`-b` or `-i`) *`pos_name`*. Otherwise the new files are placed at the end.

Example

To add or replace specified files in a library, use:

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

To replace files in a library, and only if the file in the library is older than the specified file, use:

```
armar -ru mylib k*.o
```

See also

- *-a pos_name* on page 3-5
- *-b pos_name* on page 3-6
- *-i pos_name* on page 3-8
- *-u* on page 3-13.

3.2.19 `--reinitialize_workdir`

Enables you to reinitialize the project template working directory set using `--workdir`.

When the directory set using `--workdir` refers to an existing working directory containing modified project template files, specifying this option causes the working directory to be deleted and recreated with new copies of the original project template files.

Restrictions

This option must be used in combination with the `--workdir` option.

See also

- *--project=filename, --no_project=filename* on page 3-10
- *--workdir=directory* on page 3-14.

3.2.20 `-s`

Creates a symbol table in the library. This option is useful for libraries that have been created:

- using the `-n` option
- with an archiver that does not automatically create a symbol table.

Note

By default, `armar` always creates a symbol table when you create a library of object files.

Example

To create a symbol table in a library that was created using the `-n` option, use:

```
armar -s mylib.a
```

See also

- *-n* on page 3-9
- *--zs* on page 3-15.

3.2.21 `--show_cmdline`

This option shows how `armar` has processed the command line. The commands are shown normalized, and the contents of any via files are expanded.

3.2.22 `--sizes`

Lists the `Code`, `RO Data`, `RW Data`, `ZI Data`, and `Debug` sizes of each member in the library.

Example

The following example shows the sizes of `app_1.o` and `app_2.o` in `mylib.a`:

```
> armar --sizes mylib.a

Code    RO Data    RW data    ZI Data    Debug   Object Name
 464          0          0          0     8612   app_1.o
3356          0          0      10244    11848   app_2.o
3820          0          0      10244    20460   TOTAL
```

See also

- *--entries* on page 3-7
- *--zt* on page 3-16.

3.2.23 `-t`

Prints a table of contents for the library. The files specified by *`file_list`* are included in the written list. If *`file_list`* is not specified, all files in the library are included in the order of the archive.

Example

To list the table of contents of a library in verbose mode, use:

```
armar -tv mylib
```

3.2.24 `-T`

Enables filename truncation of extracted files whose library names are longer than the file system can support. By default, extracting a file with a name that is too long is an error. A diagnostic message is written and the file is not extracted.

3.2.25 `-u`

Updates older files. When used with the `-r` option, files in the library are replaced only if the corresponding file has a modification time that is at least as new as the modification time of the file within library.

See also

- *--new_files_only* on page 3-9
- *-r* on page 3-11.

3.2.26 `-v`

Gives verbose output.

The output depends on what other options are used:

`-d, -r, -x`
: Write a detailed file-by-file description of the library creation, the constituent files, and maintenance activity.

`-p`
: Writes the name of the file to the standard output before writing the file itself to the `stdout`.

`-t`
: Includes a long listing of information about the files within the library.

`-x`
: Prints the filename preceding each extraction.

See also

- *-d* on page 3-6
- *-p* on page 3-9
- *-r* on page 3-11
- *-t* on page 3-13
- *-x* on page 3-15.

3.2.27 `--via=`*`file`*

Instructs the librarian to use options specified in *`file`*.

See also

- Appendix A *Via File Syntax* in the *Compiler Reference Guide*.

3.2.28 `--vsn`

Prints the version number to `stderr`.

See also

- *--help* on page 3-8.

3.2.29 `--workdir=`*`directory`*

Enables you to provide a working directory for a project template.

Note

Project templates only require working directories if they include files, for example, RealView Debugger configuration files.

Syntax

```
--workdir=directory
```

Where `directory` is the name of the project directory.

Restrictions

If you specify a project working directory using `--workdir`, then you must specify a project file using `--project`.

Errors

An error message is produced if you try to use `--project` without `--workdir` and `--workdir` is required.

See also

- *--project=filename, --no_project=filename* on page 3-10
- *--reinitialize_workdir* on page 3-11.

3.2.30 `-x`

Extracts the files in `file_list` from the library. The contents of the library are not changed. If no file operands are given, all files in the library are extracted. If the filename of a file extracted from the library is longer than that supported in the destination directory, the results are undefined.

3.2.31 `--zs`

Displays the symbol table for all files in the library.

Example

To list the symbol table in a library, use:

```
armar --zs mylib
```

See also

- *-n* on page 3-9
- *-s* on page 3-12.

Copyright © 2008-2010 ARM. All rights reserved.
Non-Confidential,

3.2.32 `--zt`

Lists both the member sizes and entry points for all files in the library. See `--sizes` and `--entries` for the output format.

See also

- *--entries* on page 3-7
- *--sizes* on page 3-12.