ARM® Compiler

Version 6.00

fromelf User Guide

ARM®

ARM DUI 0805A (ID031214)

ARM Compiler
fromelf User Guide

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
14 March 2014	A	Non-Confidential	ARM Compiler v6.00 Release

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM Compiler fromelf User Guide

Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

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

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

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

`monospace bold`

Denotes language keywords when used outside example code.

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.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, 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 content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0805A
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center `http://infocenter.arm.com/help/index.jsp`
- ARM Technical Support Knowledge Articles `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html`
- ARM Support and Maintenance `http://www.arm.com/support/services/support-maintenance.php`
- ARM Glossary `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
Overview of the fromelf image converter

The following topics give an overview of the `fromelf` image converter provided with the ARM Compiler toolchain:

Tasks

- *Getting help on the fromelf command* on page 2-5

Concepts

- *About the fromelf image converter* on page 2-2
- *Considerations when using fromelf* on page 2-4.

Reference

- *fromelf command-line syntax* on page 2-6
- *fromelf command-line options listed in groups* on page 2-7.

2.1 About the fromelf image converter

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 (AArch32 state only)
  - Intel Hex-32 (AArch32 state only)
  - 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.

2.1.1 See also

Concepts

- *fromelf execution modes* on page 2-3
- *Considerations when using fromelf* on page 2-4
- *Protecting code in images and objects with fromelf* on page 3-9.

Reference

- *fromelf command-line syntax* on page 2-6
- *fromelf command-line options listed in groups* on page 2-7

2.2 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`).

2.2.1 See also

Reference

- *--bin* on page 4-5
- *--elf* on page 4-28
- *--i32 (32-bit only)* on page 4-42
- *--m32 (32-bit only)* on page 4-54
- *--text* on page 4-71
- *--vhx* on page 4-74.

2.3 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.

  — A file is not created for a load region if all the execution regions within that load region are empty.

  If you convert an ELF image built for AArch32 state that contains multiple load regions, and you use either the `--m32combined` or `--i32combined` option, `fromelf`:

  1. Creates an output directory named *`destination`*.
  2. Generates one binary output file for all load regions in the input image.
  3. Places the output file in the *`destination`* directory.

  ELF images contain multiple load regions if, for example, they are built with a scatter 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.

2.3.1 See also

Reference

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--bin* on page 4-5
- *--i32 (32-bit only)* on page 4-42
- *--i32combined (32-bit only)* on page 4-43
- *--m32 (32-bit only)* on page 4-54
- *--m32combined (32-bit only)* on page 4-55
- *--vhx* on page 4-74.

2.4 Getting help on the fromelf command

Use the `--help` option to display a summary of the main command-line options.

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

2.4.1 Example

To display the help information, enter:

```
fromelf --help
```

2.4.2 See also

Reference

- *fromelf command-line syntax* on page 2-6
- *--help* on page 4-39.

2.5 fromelf command-line syntax

The `fromelf` command-line syntax is:

`fromelf [options] input_file`

options fromelf command-line options.

input_file The ELF file or library file to be processed. When some options are used, multiple input files can be specified.

2.5.1 See also

Reference

- *fromelf command-line options listed in groups* on page 2-7
- *input_file* on page 4-48.

2.6 fromelf command-line options listed in groups

The fromelf command-line options are:

Controlling the output format of build attributes

- *--decode_build_attributes* on page 4-19
- *--dump_build_attributes* on page 4-27
- *--extract_build_attributes* on page 4-32.

Controlling debug information in output files

- *--debugonly* on page 4-18
- *--emit=option[,option,...]* on page 4-29
- *--strip=option[,option,...]* on page 4-68.

Controlling diagnostic information in output files

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

- *--compare=option[,option,...]* on page 4-12
- *--continue_on_error* on page 4-14
- *--diag_error=tag[,tag,...]* on page 4-21
- *--diag_remark=tag[,tag,...]* on page 4-22
- *--diag_style={arm|ide|gnu}* on page 4-23
- *--diag_suppress=tag[,tag,...]* on page 4-24
- *--diag_warning=tag[,tag,...]* on page 4-25
- *--ignore_section=option[,option,...]* on page 4-44
- *--ignore_symbol=option[,option,...]* on page 4-45
- *--relax_section=option[,option,...]* on page 4-60
- *--relax_symbol=option[,option,...]* on page 4-61
- *--show_cmdline* on page 4-66.

Command-line help

- *--help* on page 4-39
- *--version_number* on page 4-73
- *--vsn* on page 4-76.

Getting command-line arguments from a file

- *--via=filename* on page 4-75.

Controlling miscellaneous factors affecting the image content

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--cad* on page 4-9
- *--cadcombined* on page 4-11
- *--cpu=list* on page 4-15
- *--cpu=name* on page 4-16
- *--disassemble* on page 4-26
- *--emit=option[,option,...]* on page 4-29
- *--expandarrays* on page 4-31
- *--fieldoffsets* on page 4-34
- *--fpu=list* on page 4-36
- *--fpu=name* on page 4-37
- *--globalize=option[,option,...]* on page 4-38
- *--hide=option[,option,...]* on page 4-40

- *--hide_and_localize=option[,option,...]* on page 4-41
- *--in_place* on page 4-46
- *--interleave=option* on page 4-50
- *--linkview, --no_linkview* on page 4-52
- *--localize=option[,option,...]* on page 4-53
- *--qualify* on page 4-59
- *--rename=option[,option,...]* on page 4-62
- *--select=select_options* on page 4-63
- *--show=option[,option,...]* on page 4-64
- *--show_and_globalize=option[,option,...]* on page 4-65
- *--source_directory=path* on page 4-67
- *--strip=option[,option,...]* on page 4-68
- *--symbolversions, --no_symbolversions* on page 4-70.

Obtaining a floating license

- *--licretry* on page 4-51.

Controlling the output format

- *--bin* on page 4-5
- *--bincombined* on page 4-6
- *--bincombined_base=address* on page 4-7
- *--bincombined_padding=size,num* on page 4-8
- *--elf* on page 4-28
- *--i32 (32-bit only)* on page 4-42
- *--i32combined (32-bit only)* on page 4-43
- *--m32 (32-bit only)* on page 4-54
- *--m32combined (32-bit only)* on page 4-55
- *--output=destination* on page 4-57
- *--vhx* on page 4-74
- *--wide64bit* on page 4-78
- *--widthxbanks* on page 4-79.

Controlling the display of information

- *--info=topic[,topic,...]* on page 4-47
- *--only=section_name* on page 4-56
- *--text* on page 4-71
- *-w* on page 4-77.

Protecting IP in images and objects

- *--privacy* on page 4-58
- *--strip=option[,option,...]* on page 4-68.

Chapter 3
Using fromelf

The following topics describe how to use the image `fromelf` conversion utility provided with the ARM Compiler toolchain:

Tasks

3.1 Converting an ELF image to Intel Hex-32 format (32-bit only)

Use one of these options to produce Intel Hex-32 format output:

- `--i32`
- `--i32combined`

`--i32` generates one output file for each load region in the image. However, a file is not created for a load region if all the execution regions within that load region are empty.

`--i32combined` generates one output file for an image containing multiple load regions.

Note

The following restrictions apply:

- these options are supported only for AArch32 state files
- you cannot use these options with object files
- you must use `--output` with these options.

You can specify the base address of the output with the `--base` option.

3.1.1 Example

To convert the ELF file `infile.axf` to an Intel Hex-32 format file, for example `outfile.bin`, enter:

```
fromelf --cpu=8-A.32 --i32 --output=outfile.bin infile.axf
```

To create a single output file,`outfile2.bin`, from an image file `infile2.axf`, with two load regions, and with a start address of `0x1000`, enter:

```
fromelf --cpu=8-A.32 --i32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

3.1.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *fromelf command-line syntax* on page 2-6
- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--i32 (32-bit only)* on page 4-42
- *--i32combined (32-bit only)* on page 4-43
- *--output=destination* on page 4-57.

3.2 Converting an ELF image to Motorola 32-bit format (32-bit only)

Use one of these options to produce Motorola 32-bit format (32-bit S-records) output:

- `--m32`
- `--m32combined`

`--m32` generates one output file for each load region in the image. However, a file is not created for a load region if all the execution regions within that load region are empty.

`--m32combined` generates one output file for an image containing multiple load regions.

Note

The following restrictions apply:

- these options are supported only for AArch32 state files
- you cannot use these options with object files
- you must use `--output` with these options.

You can specify the base address of the output with the `--base` option.

3.2.1 Example

To convert the ELF file `infile.axf` to a Motorola 32-bit format file, for example `outfile.bin`, enter:

```
fromelf --cpu=8-A.32 --m32 --output=outfile.bin infile.axf
```

To create a single Motorola 32-bit format output file, `outfile2.bin`, from an image file `infile2.axf`, with two load regions, and with a start address of `0x1000`, enter:

```
fromelf --cpu=8-A.32 --m32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

3.2.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *fromelf command-line syntax* on page 2-6
- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--m32 (32-bit only)* on page 4-54
- *--m32combined (32-bit only)* on page 4-55
- *--output=destination* on page 4-57.

3.3 Converting an ELF image to plain binary format

Use the `--bin` option to produce plain binary output, one file for each load region. However, a file is not created for a load region if all the execution regions within that load region are empty.

You can split the output from this option into multiple files with the *`--widthxbanks`* option.

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

Be aware of the following when using these options:

- You must use the `--output` option with `--bin` and `--bincombined`.
- For `--bincombined`, if you use a scatter file that defines two load regions with a large address space between them, the resulting binary can be very large because it contains mostly padding. For example, if you have a load region of size `0x100` bytes at address `0x00000000` and another load region at address `0x30000000`, the amount of padding is `0x2FFFFF00` bytes.

3.3.1 Examples

To convert an ELF file to a plain binary file, for example `outfile.bin`, enter:

```
fromelf --cpu=8-A.64 --bin --output=out.bin in.axf
```

To produce a binary file that can be loaded at start address `0x1000`, enter:

```
fromelf --cpu=8-A.64 --bincombined --bincombined_base=0x1000 --output=out.bin in.axf
```

To produce plain binary output and fill the space between load regions with copies of the 32-bit word `0x12345678`, enter:

```
fromelf --cpu=8-A.64 --bincombined --bincombined_padding=4,0x12345678 --output=out.bin
in.axf
```

3.3.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *fromelf command-line syntax* on page 2-6
- *--bin* on page 4-5
- *--bincombined* on page 4-6
- *--bincombined_base=address* on page 4-7
- *--bincombined_padding=size,num* on page 4-8
- *--output=destination* on page 4-57
- *--widthxbanks* on page 4-79.

3.4 Converting an ELF image to Byte oriented (Verilog Memory Model) hexadecimal format

Use the `--vhx` option to produce 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.

Note

You must use `--output` with these options.

3.4.1 Examples

To convert the ELF file `infile.axf` to a byte oriented hexadecimal format file, for example `outfile.bin`, enter:

```
fromelf --cpu=8-A.64 --vhx --output=outfile.bin infile.axf
```

To create multiple output files, in the `regions` directory, from an image file multiload.axf, with two 8-bit memory banks, enter:

```
fromelf --cpu=8-A.64 --vhx --8x2 multiload.axf --output=regions
```

3.4.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *fromelf command-line syntax* on page 2-6
- *--output=destination* on page 4-57
- *--vhx* on page 4-74
- *--widthxbanks* on page 4-79.

3.5 Controlling debug information in output files

Use the `--debugonly` option to remove 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.

Note

You must use `--elf` with this option. Because you have to use `--elf`, you must also use `--output`.

3.5.1 Example

To create an ELF file, `debugout.axf`, from the ELF file `infile.axf`, containing only debug information, enter:

```
fromelf --cpu=8-A.64 --elf --debugonly --output=debugout.axf infile.axf
```

3.5.2 See also

Reference

- *fromelf command-line syntax* on page 2-6
- *--debugonly* on page 4-18
- *--elf* on page 4-28
- *--output=destination* on page 4-57.

3.6 Disassembling an ELF-formatted file

Use the `--disassemble` option to display a disassembled version of the image to `stdout`. If you use this option with the `--output` *destination* option, 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`.

3.6.1 Example

To disassemble the ELF file `infile.axf` for the 8-A.32 target and create a source file `outfile.asm`, enter:

```
fromelf --cpu=8-A.32 --disassemble --output=outfile.asm infile.axf
```

3.6.2 See also

Reference

- *fromelf command-line syntax* on page 2-6
- *--cpu=name* on page 4-16
- *--disassemble* on page 4-26
- *--emit=option[,option,...]* on page 4-29
- *--interleave=option* on page 4-50
- *--output=destination* on page 4-57
- *--text* on page 4-71.

armasm User Guide:

- Chapter 9 *Using armasm*.

3.7 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
```

3.7.1 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 --cpu=8-A.64 --elf --strip=all test.a -o strip_all/
```

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

3.7.2 Example of processing a subset of files in the archive

To remove all debug, comments, notes and symbols from only the `shapes.o` and the `strmtst.o` files in the archive, enter:

```
fromelf --cpu=8-A.64 --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.

To process the `bmw.o`, `bmw1.o`, and `newtst.o` files in the archive, enter:

```
fromelf --cpu=8-A.64 --elf --strip=all test.a(??w*) -o subset/
```

3.7.3 Example of displaying a disassembled version of files in an archive

To display the disassembled version of `call_c_code.o` in the archive, enter:

```
fromelf --cpu=8-A.64 --disassemble test.a(c*)
```

3.7.4 See also

Reference

- *--disassemble* on page 4-26
- *--elf* on page 4-28
- *input_file* on page 4-48
- *--output=destination* on page 4-57
- *--strip=option[,option,...]* on page 4-68.

3.8 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`.

3.8.1 Protecting code in image files

For image files:

Table 3-1 Effect of fromelf --privacy and --strip options on images files

Option [a]	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

a. The `--cpu` option omitted for brevity.

3.8.2 Protecting code in object files

For object files:

Table 3-2 Effect of fromelf --privacy and --strip options on object files

Option [a]	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

a. The `--cpu` option omitted for brevity.

3.8.3 Example

To produce a new ELF executable image with the complete symbol table removed and various section names changed, enter:

```
fromelf --cpu=8-A.64 --elf --privacy --output=outfile.axf infile.axf
```

3.8.4 See also

Concepts

armlink User Guide:

- *About mapping symbols* on page 7-3.

Reference

- *fromelf command-line syntax* on page 2-6
- *--elf* on page 4-28
- *--output=destination* on page 4-57
- *--privacy* on page 4-58
- *--strip=option[,option,...]* on page 4-68.

armlink Reference Guide:

- *--list_mapping_symbols, --no_list_mapping_symbols* on page 2-82
- *--locals, --no_locals* on page 2-84

- --*privacy* on page 2-101.

3.9 Printing details of ELF-formatted files

You can specify the elements of an ELF object that you want to appear in the textual output with the `--emit` option. The output includes ELF header and section information. You can specify these elements as a comma separated list.

Note

You can specify some of the `--emit` options using the `--text` option.

3.9.1 Example of printing data sections

To print the contents of the data sections of an ELF file, `infile.axf`, enter:

```
fromelf --cpu=8-A.64 --emit=data infile.axf
```

3.9.2 Example of printing relocation information

To print relocation information and the dynamic section contents for the ELF file `infile2.axf`, enter:

```
fromelf --cpu=8-A.64 --emit=relocation_tables,dynamic_segment infile2.axf
```

3.9.3 See also

Reference

- *fromelf command-line syntax* on page 2-6
- *--emit=option[,option,...]* on page 4-29
- *--text* on page 4-71.

3.10 Using fromelf to find where a symbol is placed in an executable ELF image

To find where a symbol is placed in an ELF image file, use the `--text -s -v` options to view the symbol table and detailed information on each segment and section header, for example:

```
fromelf --text -s -v s.axf
```

The symbol table identifies the section where the symbol is placed.

3.10.1 Example

Do the following:

1. Create the file `s.c` containing the following source code:

   ```
   long long altstack[10] __attribute__ ((section ("STACK"), zero_init));

   int main()
   {
       return sizeof(altstack);
   }
   ```

2. Compile the source:

   ```
   armclang -target armv8a-arm-none-eabi -c s.c -o s.o
   ```

3. Link the object `s.o` and keep the `STACK` symbol:

   ```
   armlink --force-scanlib --cpu=8-A.32 --keep=s.o(STACK) s.o --output=s.axf
   ```

4. Run the fromelf command to display the symbol table and detailed information on each segment and section header:

   ```
   fromelf --cpu=8-A.32 --text -s -v s.axf
   ```

5. Locate the `STACK` and `altstack` symbols in the `fromelf` output, for example:

   ```
   ...
   ** Section #4

       Name        : .symtab
       Type        : SHT_SYMTAB (0x00000002)
       Flags       : None (0x00000000)
       Addr        : 0x00000000
       File Offset : 776 (0x308)
       Size        : 2816 bytes (0xb00)
       Link        : Section 5 (.strtab)
       Info        : Last local symbol no = 111
       Alignment   : 4
       Entry Size  : 16

       Symbol table .symtab (175 symbols, 111 local)

         #  Symbol Name                Value      Bind  Sec  Type  Vis  Size
       ==========================================================================
       ...
        13  STACK                      0x00008200   Lc    2  Sect  De   0x50
       ...

       174  altstack                   0x00008200   Gb    2  Data  Hi   0x50
       ...
   ```

   The `Sec` column shows the section where the stack is placed. In this example, section 4.

6. Locate the section identified for the symbol in the `fromelf` output, for example:

```
...
==================================

** Section #2

    Name         : ER_ZI
    Type         : SHT_NOBITS (0x00000008)
    Flags        : SHF_ALLOC + SHF_WRITE (0x00000003)
    Addr         : 0x0000819c
    File Offset : 464 (0x1d0)
    Size         : 180 bytes (0xb4)
    Link         : SHN_UNDEF
    Info         : 0
    Alignment    : 8
    Entry Size   : 0

==================================
...
```

This shows that the symbols are placed in a ZI execution region.

3.10.2 See also

Tasks

- *How to find where a symbol is placed when linking* on page 6-6.

Reference

- *--text* on page 4-71.

armlink Reference Guide:

- *--keep* on page 2-70
- *--output* on page 2-92.

Chapter 4
fromelf command reference

The following topics describe the command-line options of the `fromelf` image conversion utility provided with the ARM Compiler toolchain:

See also *fromelf command-line syntax* on page 2-6.

4.1 --base [[*object_file*::]*load_region_ID*=]*num* (32-bit only)

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.

4.1.1 Restrictions

You must use one of the output formats --i32, --i32combined, --m32, or --m32combined with this option. Therefore, you cannot use this option with object files.

Note

This option is supported only for AArch32 state files.

4.1.2 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.

You can:

- use wildcard characters ? and * for symbolic names in *object_file* and *load_region_ID* arguments
- specify multiple options in one --base option followed by a comma-separated list of arguments.

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 4-1 Examples of 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

4.1.3 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--i32 (32-bit only)* on page 4-42

- *--i32combined (32-bit only)* on page 4-43
- *--m32 (32-bit only)* on page 4-54
- *--m32combined (32-bit only)* on page 4-55.

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 *--widthxbanks* option.

4.2.1 Restrictions

You cannot use this option with object files.

You must use --output with this option.

4.2.2 Considerations when using --bin

A file is not created for a load region if all the execution regions within that load region are empty.

4.2.3 Example

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

```
fromelf --cpu=8-A.32 --bin --output=outfile.bin infile.axf
```

4.2.4 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--output=destination* on page 4-57
- *--widthxbanks* on page 4-79.

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.

4.3.1 Restrictions

You cannot use this option with object files.

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

4.3.2 Considerations when using --bincombined

Be aware of the following:

- Use this option with `--bincombined_base` to change the default value for the base address.
- The default padding value is `0xFF`. Use this option with `--bincombined_padding` to change the default padding value.
- If you use a scatter file that defines two load regions with a large address space between them, the resulting binary can be very large because it contains mostly padding. For example, if you have a load region of size `0x100` bytes at address `0x00000000` and another load region at address `0x30000000`, the amount of padding is `0x2FFFFF00` bytes.
- ARM recommends that you use a different method of placing widely spaced load regions, such as splitting the binary file into multiple files with the *`--widthxbanks`* option.

4.3.3 See also

Concepts

armlink User Guide:

- *Input sections, output sections, regions, and Program Segments* on page 4-5.

Reference

- *--bincombined_base=address* on page 4-7
- *--bincombined_padding=size,num* on page 4-8
- *--output=destination* on page 4-57
- *--widthxbanks* on page 4-79.

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.

4.4.1 Restrictions

You must use `--bincombined` with this option. If you omit `--bincombined`, a warning message is displayed.

4.4.2 Syntax

`--bincombined_base=`*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.

4.4.3 Default

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

4.4.4 Example

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

4.4.5 See also

Concepts

armlink User Guide:

- *Input sections, output sections, regions, and Program Segments* on page 4-5.

Reference

- *--bincombined* on page 4-6
- *--bincombined_padding=size,num* on page 4-8.

4.5 --bincombined_padding=*size*,*num*

This option enables you to specify a different padding value from the default used by the --bincombined output mode.

4.5.1 Restrictions

You must use --bincombined with this option. If you omit --bincombined, a warning message is displayed.

4.5.2 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. If you specify a value that is too large to fit in the specified size, a warning message is displayed.

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.

4.5.3 Default

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

4.5.4 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.

4.5.5 See also

Reference

- *--bincombined* on page 4-6
- *--bincombined_base=address* on page 4-7.

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.

4.6.1 Restrictions

You cannot use this option with object files.

4.6.2 Considerations when using --cad

A file is not created for a load region if all the execution regions within that load region are empty.

4.6.3 Example

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

- To produce an array definition in the file `load_region.c` for an image that has a single load region, enter:

  `fromelf --cpu=8-A.32 --cad myimage.axf --output load_region.c`

  The file contains, for example:

```
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,
    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.

4.6.4 See also

Tasks

armlink User Guide:

- Chapter 8 *Using scatter files*.

Concepts

armlink User Guide:

- *Input sections, output sections, regions, and Program Segments* on page 4-5.

Reference

- *--cadcombined* on page 4-11
- *--output=destination* on page 4-57.

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.

The output is directed to `stdout` by default. To save the output to a file, use the `--output` option together with a filename.

4.7.1 Restrictions

You cannot use this option with object files.

4.7.2 Example

The following commands create the file `load_regions.c` in the directory *root*`\myprojects\multiload`:

```
cd root\myprojects\multiload fromelf --cpu=8-A.32 --cadcombined image_multiload.axf
--output load_regions.c
```

4.7.3 See also

Tasks

armlink User Guide:

- Chapter 8 *Using scatter files*.

Reference

- *--cad* on page 4-9
- *--output=destination* on page 4-57.

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.

4.8.1 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*.

You can:

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

4.8.2 See also

Reference

- *--ignore_section=option[,option,...]* on page 4-44
- *--ignore_symbol=option[,option,...]* on page 4-45
- *--relax_section=option[,option,...]* on page 4-60
- *--relax_symbol=option[,option,...]* on page 4-61.

4.9 --continue_on_error

This option reports any errors and then continues.

Use `--diag_warning=error` instead of this option.

4.9.1 See also

Reference

- *--diag_warning=tag[,tag,...]* on page 4-25.

4.10 --cpu=list

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

4.10.1 See also

Reference

- *--cpu=name* on page 4-16.

4.11 --cpu=*name*

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

4.11.1 Syntax

`--cpu=`*`name`*

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

4.11.2 Example

To select the disassembly for 8-A.32, use:

```
--cpu=8-A.32 --disassemble
```

4.11.3 See also

Reference

- *--cpu=list* on page 4-15
- *--disassemble* on page 4-26
- *--info=topic[,topic,...]* on page 4-47
- *--text* on page 4-71.

armasm Reference Guide:

- *--cpu* on page 2-15.

armlink Reference Guide:

- *--cpu* on page 2-29.

4.12 --datasymbols

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

You can use this option only with `--text -d`.

4.12.1 See also

Reference

- *--text* on page 4-71.

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.

4.13.1 Restrictions

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

4.13.2 See also

Reference

- *--elf* on page 4-28.

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*.

4.14.1 Restrictions

You can use this option only in text mode for 8-A.32 targets.

This option has no effect for 8-A.64 targets.

4.14.2 Example

The following example shows the output for --decode_build_attributes:

```
** Section #10 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 89 bytes

    'aeabi' file build attributes:
    0x000000:   43 32 2e 30 36 00 05 38 2d 41 2e 33 32 00 06 0a    C2.06..8-A.32...
    0x000010:   07 41 08 01 09 02 0a 05 0c 02 11 01 12 02 14 02    .A..............
    0x000020:   17 01 18 01 19 01 1a 01 1c 01 1e 03 22 01 24 01    ............".$.
    0x000030:   42 01 44 03 46 01 2c 02                            B.D.F.,.
        Tag_conformance = "2.06"
        Tag_CPU_name = "8-A.32"
        Tag_CPU_arch = ARM v7 (=10)
        Tag_CPU_arch_profile = The application profile 'A' (e.g. for Cortex A8) (=65)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_THUMB_ISA_use = Thumb2 instructions were permitted (implies Thumb instructions permitted) (=2)
        Tag_VFP_arch = VFPv4 instructions were permitted (implies VFPv3 instructions were permitted) (=5)
        Tag_NEON_arch = Use of Advanced SIMD Architecture version 2 was permitted (=2)
        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_VFP_args = FP parameter/result passing conforms to the VFP variant of the AAPCS (=1)
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion preserved (=3)
        Tag_CPU_unaligned_access = The producer was permitted to generate architecture v6-style unaligned data
accesses (=1)
        Tag_VFP_HP_extension = The producer was permitted to use the VFPv3/Advanced SIMD optional half-precision
extension (=1)
        Tag_T2EE_use = Use of the T2EE extension was permitted (=1)
        Tag_Virtualization_use = Use of TrustZone and virtualization extensions was permitted (=3)
        Tag_MPextension_use = Use of the ARM v7 MP extension was permitted (=1)
        Tag_v7DIV_use = Code was permitted to use SDIV and UDIV; code is intended to execute on a CPU conforming
to architecture v7 with the integer division extension (=2)

    'ARM' file build attributes:
    0x000000:   12 01 16 01                                        ....
```

4.14.3 See also

Reference

- *--dump_build_attributes* on page 4-27
- *--emit=option[,option,...]* on page 4-29
- *--extract_build_attributes* on page 4-32.

Other information

- *Application Binary Interface for the ARM Architecture*
  http://infocenter.arm.com/help/topic/com.arm.doc.ihi0036-/index.html.

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

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

4.15.1 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.

4.15.2 See also

Reference

- *--diag_remark=tag[,tag,...]* on page 4-22
- *--diag_style={arm|ide|gnu}* on page 4-23
- *--diag_suppress=tag[,tag,...]* on page 4-24
- *--diag_warning=tag[,tag,...]* on page 4-25.

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

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

4.16.1 Syntax

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

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

4.16.2 See also

Reference

- *--diag_error=tag[,tag,...]* on page 4-21
- *--diag_style={arm|ide|gnu}* on page 4-23
- *--diag_suppress=tag[,tag,...]* on page 4-24
- *--diag_warning=tag[,tag,...]* on page 4-25.

4.17 --diag_style={arm|ide|gnu}

This option specifies the style to use for diagnostic messages.

4.17.1 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.

4.17.2 Default

The default is `--diag_style=arm`.

4.17.3 See also

Reference

- *--diag_error=tag[,tag,...]* on page 4-21
- *--diag_remark=tag[,tag,...]* on page 4-22
- *--diag_suppress=tag[,tag,...]* on page 4-24
- *--diag_warning=tag[,tag,...]* on page 4-25.

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

This option disables diagnostic messages that have the specified tags.

4.18.1 Syntax

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

Where *tag* can be:

- a diagnostic message number to be suppressed
- `error`, to suppress all errors
- `warning`, to suppress all warnings.

4.18.2 See also

Reference

- *--diag_error=tag[,tag,...]* on page 4-21
- *--diag_remark=tag[,tag,...]* on page 4-22
- *--diag_style={arm|ide|gnu}* on page 4-23
- *--diag_warning=tag[,tag,...]* on page 4-25.

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

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

4.19.1 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.

4.19.2 See also

Reference

- *--diag_error=tag[,tag,...]* on page 4-21
- *--diag_remark=tag[,tag,...]* on page 4-22
- *--diag_style={arm|ide|gnu}* on page 4-23
- *--diag_warning=tag[,tag,...]*.

4.20 --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`.

4.20.1 See also

Reference

- *--cpu=name* on page 4-16
- *--emit=option[,option,...]* on page 4-29
- *--interleave=option* on page 4-50
- *--output=destination* on page 4-57
- *--text* on page 4-71.

4.21 --dump_build_attributes

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

4.21.1 Restrictions

You can use this option only in text mode for 8-A.32 targets.

This option has no effect for 8-A.64 targets.

4.21.2 Example

The following example shows the output for --dump_build_attributes:

```
...
** Section #10 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 89 bytes

    0x000000:   41 47 00 00 00 61 65 61 62 69 00 01 3d 00 00 00    AG...aeabi..=...
    0x000010:   43 32 2e 30 36 00 05 38 2d 41 2e 33 32 00 06 0a    C2.06..8-A.32...
    0x000020:   07 41 08 01 09 02 0a 05 0c 02 11 01 12 02 14 02    .A..............
    0x000030:   17 01 18 01 19 01 1a 01 1c 01 1e 03 22 01 24 01    ............".$.
    0x000040:   42 01 44 03 46 01 2c 02 11 00 00 00 41 52 4d 00    B.D.F.,.....ARM.
    0x000050:   01 09 00 00 00 12 01 16 01                         .........
```

4.21.3 See also

Reference

- *--decode_build_attributes* on page 4-19
- *--emit=option[,option,...]* on page 4-29
- *--extract_build_attributes* on page 4-32
- *--text* on page 4-71.

4.22 --elf

This option selects ELF output mode.

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

4.22.1 Restrictions

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

4.22.2 See also

Reference

- *--in_place* on page 4-46
- *--output=destination* on page 4-57
- *--strip=option[,option,...]* on page 4-68.

4.23 --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.

4.23.1 Restrictions

You can use this option only in text mode.

4.23.2 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. The produces the same output as the `--decode_build_attributes` option.

`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 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.

4.23.3 See also

Reference

- *--disassemble* on page 4-26
- *--decode_build_attributes* on page 4-19
- *--expandarrays* on page 4-31
- *--text* on page 4-71.

4.24 --expandarrays

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

4.24.1 Restrictions

You can use this option only with `--text -a`.

4.24.2 See also

Reference

- *--text* on page 4-71.

4.25 --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.

4.25.1 Restrictions

You can use this option only in text mode for 8-A.32 targets.

This option has no effect for 8-A.64 targets.

4.25.2 Example

The following example shows the output for --extract_build_attributes:

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

** Object/Image Build Attributes

    'aeabi' file build attributes:
    0x000000:   43 32 2e 30 36 00 05 38 2d 41 2e 33 32 00 06 0a    C2.06..8-A.32...
    0x000010:   07 41 08 01 09 02 0a 05 0c 02 11 01 12 02 14 02    .A..............
    0x000020:   17 01 18 01 19 01 1a 01 1c 01 1e 03 22 01 24 01    ............".$.
    0x000030:   42 01 44 03 46 01 2c 02                            B.D.F.,.
        Tag_conformance = "2.06"
        Tag_CPU_name = "8-A.32"
        Tag_CPU_arch = ARM v7 (=10)
        Tag_CPU_arch_profile = The application profile 'A' (e.g. for Cortex A8) (=65)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_THUMB_ISA_use = Thumb2 instructions were permitted (implies Thumb instructions permitted) (=2)
        Tag_VFP_arch = VFPv4 instructions were permitted (implies VFPv3 instructions were permitted) (=5)
        Tag_NEON_arch = Use of Advanced SIMD Architecture version 2 was permitted (=2)
        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_VFP_args = FP parameter/result passing conforms to the VFP variant of the AAPCS (=1)
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion preserved (=3)
        Tag_CPU_unaligned_access = The producer was permitted to generate architecture v6-style unaligned data
accesses (=1)
        Tag_VFP_HP_extension = The producer was permitted to use the VFPv3/Advanced SIMD optional half-precision
extension (=1)
        Tag_T2EE_use = Use of the T2EE extension was permitted (=1)
        Tag_Virtualization_use = Use of TrustZone and virtualization extensions was permitted (=3)
        Tag_MPextension_use = Use of the ARM v7 MP extension was permitted (=1)
        Tag_v7DIV_use = Code was permitted to use SDIV and UDIV; code is intended to execute on a CPU conforming
to architecture v7 with the integer division extension (=2)

    'ARM' file build attributes:
    0x000000:   12 01 16 01                                        ....
```

4.25.3 See also

Reference

- *--decode_build_attributes* on page 4-19

- *--dump_build_attributes* on page 4-27
- *--emit=option[,option,...]* on page 4-29
- *--text* on page 4-71.

4.26 --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.

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.

4.26.1 Restrictions

This option:

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

4.26.2 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, enter:

  ```
  fromelf --cpu=8-A.32 --fieldoffsets inputfile.o
  ```

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

  ```
  fromelf --cpu=8-A.32 --fieldoffsets --select=p* --output=outputfile.s inputfile.o
  ```

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

  ```
  fromelf --cpu=8-A.32 --fieldoffsets --select=tools.*,moretools.*
  --output=outputfile.s inputfile.o
  ```

- To produce an output file listing to outputfile.s 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, enter:

  ```
  fromelf --cpu=8-A.32 --fieldoffsets --select=tools.top.number*
  --output=outputfile.s inputfile.o
  ```

The following is an example of the output:

```
; Structure, Table , Size 0x104 bytes, from inputfile.cpp
|Table.TableSize|                        EQU    0        ;  int
|Table.Data|                             EQU    0x4      ;  array[64] of MyClassHandle
; End of Structure Table

; Structure, Box2 , Size 0x8 bytes, from inputfile.cpp
|Box2.|                                  EQU    0        ;  anonymous
|Box2..|                                 EQU    0        ;  anonymous
|Box2...Min|                             EQU    0        ;  Point2
|Box2...Min.x|                           EQU    0        ;  short
|Box2...Min.y|                           EQU    0x2      ;  short
```

```
|Box2...Max|                              EQU    0x4      ;  Point2
|Box2...Max.x|                            EQU    0x4      ;  short
|Box2...Max.y|                            EQU    0x6      ;  short
; Warning: duplicate name (Box2..) present in (inputfile.cpp) and in (inputfile.cpp)
; please use the --qualify option
|Box2..|                                  EQU    0        ;  anonymous
|Box2...Left|                             EQU    0        ;  unsigned short
|Box2...Top|                              EQU    0x2      ;  unsigned short
|Box2...Right|                            EQU    0x4      ;  unsigned short
|Box2...Bottom|                           EQU    0x6      ;  unsigned short
; End of Structure Box2

; Structure, MyClassHandle , Size 0x4 bytes, from inputfile.cpp
|MyClassHandle.Handle|                    EQU    0        ;  pointer to MyClass
; End of Structure MyClassHandle

; Structure, Point2 , Size 0x4 bytes, from defects.cpp
|Point2.x|                                EQU    0        ;  short
|Point2.y|                                EQU    0x2      ;  short
; End of Structure Point2

; Structure, __fpos_t_struct , Size 0x10 bytes, from C:\Program
Files\DS-5\bin\..\include\stdio.h
|__fpos_t_struct.__pos|                   EQU    0        ;  unsigned long long
|__fpos_t_struct.__mbstate|               EQU    0x8      ;  anonymous
|__fpos_t_struct.__mbstate.__state1|      EQU    0x8      ;  unsigned int
|__fpos_t_struct.__mbstate.__state2|      EQU    0xc      ;  unsigned int
; End of Structure __fpos_t_struct

    END
```

4.26.3 See also

Reference

- *--qualify* on page 4-59
- *--select=select_options* on page 4-63
- *--text* on page 4-71.

armasm Reference Guide:

- *EQU* on page 10-36
- *GET or INCLUDE* on page 10-57.

4.27 --fpu=list

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

4.27.1 See also

Reference

- *--fpu=name* on page 4-37.

4.28 --fpu=*name*

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

4.28.1 Syntax

`--fpu=`*`name`*

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

4.28.2 Example

To select disassembly for the VFPv4 architecture, use:

```
--fpu=VFPv4
```

4.28.3 See also

Reference

- *--disassemble* on page 4-26
- *--fpu=list* on page 4-36
- *--info=topic[,topic,...]* on page 4-47
- *--text* on page 4-71.

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

This option converts the selected symbols to global symbols.

4.29.1 Restrictions

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

4.29.2 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.

You can:

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

4.29.3 See also

Reference

- *--elf* on page 4-28
- *--hide=option[,option,...]* on page 4-40.

4.30 --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.

4.30.1 See also

Reference

- *--show_cmdline* on page 4-66
- *--version_number* on page 4-73
- *--vsn* on page 4-76.

4.31 --hide=*option*[,*option*,...]

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

4.31.1 Restrictions

You must use --elf with this option.

4.31.2 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*.

You can:

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

4.31.3 See also

Reference

- *--elf* on page 4-28
- *--show=option[,option,...]* on page 4-64.

4.32 --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.

4.32.1 Restrictions

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

4.32.2 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.

You can:

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

4.32.3 See also

Reference

- *--elf* on page 4-28.

4.33 `--i32` (32-bit only)

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.

4.33.1 Restrictions

The following restrictions apply:

- the option is supported only for AArch32 state files
- you cannot use this option with object files
- you must use `--output` with this option.

4.33.2 Considerations when using `--i32`

A file is not created for a load region if all the execution regions within that load region are empty.

4.33.3 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--i32combined (32-bit only)* on page 4-43
- *--output=destination* on page 4-57.

4.34 `--i32combined` (32-bit only)

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.

4.34.1 Restrictions

The following restrictions apply:

- the option is supported only for AArch32 state files
- you cannot use this option with object files
- you must use `--output` with this option.

4.34.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--i32 (32-bit only)* on page 4-42
- *--output=destination* on page 4-57.

4.35 --ignore_section=*option*[,*option*,...]

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

4.35.1 Restrictions

You must use --compare with this option.

4.35.2 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*.

You can:

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

4.35.3 See also

Reference

- *--compare=option[,option,...]* on page 4-12
- *--ignore_symbol=option[,option,...]* on page 4-45
- *--relax_section=option[,option,...]* on page 4-60.

4.36 --ignore_symbol=*option*[,*option*,...]

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

4.36.1 Restrictions

You must use --compare with this option.

4.36.2 Syntax

--ignore_symbol=*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 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 --ignore_symbol option followed by a comma-separated list of arguments.

4.36.3 See also

Reference

- *--compare=option[,option,...]* on page 4-12
- *--ignore_section=option[,option,...]* on page 4-44
- *--relax_symbol=option[,option,...]* on page 4-61.

4.37 --in_place

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

4.37.1 Restrictions

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

4.37.2 Example

To remove debug information from members of the library file `test.a`, enter:

```
fromelf --cpu=8-A.32 --elf --in_place --strip=debug test.a
```

4.37.3 See also

Reference

- *--elf* on page 4-28
- *--strip=option[,option,...]* on page 4-68.

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

This option prints information about specific topics.

4.38.1 Restrictions

You can use this option only in text mode.

4.38.2 Syntax

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

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

`instruction_usage`

Categorizes and lists the A32 and T32 instructions defined in the code sections of each input file.

Note

Supported only for AArch32 state files.

`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 A32 or T32 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 A32 or T32 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`.

4.38.3 See also

Reference

- *--text* on page 4-71.

4.39 *input_file*

This option specifies the ELF file or archive containing ELF files 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`.

4.39.1 Usage

If *input_file* is a scatter-loaded image that contains more than one load region and the output format is one of `--bin`, `--cad`, `--m32`, `--i32`, or `--vhx`, then `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 one of `--cadcombined`, `--m32combined`, or `--i32combined`, then `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.

4.39.2 Example

To strip debug information from all files in the archive beginning with s, and create a new archive, `my_archive.a`, containing the processed and unprocessed files, enter:

```
fromelf --cpu=8-A.32 --elf --strip=debug archive.a(s*.o) --output=my_archive.a
```

4.39.3 See also

Tasks

- *Processing ELF files in an archive* on page 3-8

Reference

- *--bin* on page 4-5
- *--cad* on page 4-9
- *--cadcombined* on page 4-11

- *--compare=option[,option,...]* on page 4-12
- *--elf* on page 4-28
- *--i32 (32-bit only)* on page 4-42
- *--i32combined (32-bit only)* on page 4-43
- *--in_place* on page 4-46
- *--m32 (32-bit only)* on page 4-54
- *--m32combined (32-bit only)* on page 4-55
- *--output=destination* on page 4-57
- *--text* on page 4-71
- *--vhx* on page 4-74.

4.40 --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`.

Use this option with `--source_directory` if you want to specify additional paths to search for source code.

4.40.1 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.

4.40.2 Default

The default is `--interleave=none`.

4.40.3 See also

Reference

- *--disassemble* on page 4-26
- *--emit=option[,option,...]* on page 4-29
- *--source_directory=path* on page 4-67
- *--text* on page 4-71.

4.41 --licretry

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

4.41.1 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 `ARMCOMPILER6_FROMELFOPT` environment variable. In this way, you do not have to modify your build files.

4.41.2 See also

Reference

armlink Reference Guide:

- *--licretry* on page 2-80.

armasm Reference Guide:

- *--licretry* on page 2-41.

Other information

- *ARM® DS-5™ License Management Guide* `http://infocenter.arm.com/help/topic/com.arm.doc.dui0577-/index.html`.

4.42 --linkview, --no_linkview

This option is deprecated.

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 section-level view 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.

4.42.1 Restrictions

The following restrictions apply:

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

4.42.2 Example

To get ELF format output for image.axf, enter:

```
fromelf --cpu=8-A.32 --no_linkview --elf image.axf --output=image_nlk.axf
```

4.42.3 See also

Reference

- *--elf* on page 4-28
- *--privacy* on page 4-58
- *--strip=option[,option,...]* on page 4-68.

armlink Reference Guide:

- *--privacy* on page 2-101.

Other information

- *System V Application Binary Interface – DRAFT – 17 December 2003* specification.

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

This option converts the selected symbols to local symbols.

4.43.1 Restrictions

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

4.43.2 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.

4.43.3 See also

Reference

- *--elf* on page 4-28
- *--hide=option[,option,...]* on page 4-40.

4.44 `--m32` (32-bit only)

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.

4.44.1 Restrictions

The following restrictions apply:

- the option is supported only for AArch32 state files
- you cannot use this option with object files
- you must use `--output` with this option.

4.44.2 Considerations when using `--m32`

A file is not created for a load region if all the execution regions within that load region are empty.

4.44.3 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--m32combined (32-bit only)* on page 4-55
- *--output=destination* on page 4-57.

4.45 `--m32combined` (32-bit only)

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.

4.45.1 Restrictions

The following restrictions apply:

- the option is supported only for AArch32 state files
- you cannot use this option with object files
- you must use `--output` with this option.

4.45.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--base [[object_file::]load_region_ID=]num (32-bit only)* on page 4-3
- *--m32 (32-bit only)* on page 4-54
- *--output=destination* on page 4-57.

4.46 --only=*section_name*

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

4.46.1 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 sections to display.

4.46.2 Example

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

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

  ```
  fromelf --cpu=8-A.32 --only=.symtab --text -s test.axf
  ```

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

  ```
  fromelf --cpu=8-A.32 --only=ER? test.axf
  ```

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

  ```
  fromelf --cpu=8-A.32 --only=HEAP --only=.*tab --text -s -t test.axf
  ```

4.46.3 See also

Reference

- *--text* on page 4-71.

4.47 --output=*destination*

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

4.47.1 Syntax

`--output=`*destination*

`--o=`*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.

4.47.2 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 --cpu=8-A.32 --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.

4.47.3 See also

Reference

- *--bin* on page 4-5
- *--elf* on page 4-28
- *--text* on page 4-71.

4.48 --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.

4.48.1 See also

Tasks

- *Protecting code in images and objects with fromelf* on page 3-9

Reference

- *--strip=option[,option,...]* on page 4-68

armlink Reference Guide:

- *--locals, --no_locals* on page 2-84
- *--privacy* on page 2-101.

4.49 --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.

If the source file is in a different location from the current location, then the source file path is also included.

4.49.1 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

4.49.2 See also

Reference

- *--fieldoffsets* on page 4-34.

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

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

4.50.1 Restrictions

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

4.50.2 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.

4.50.3 See also

Reference

- *--compare=option[,option,...]* on page 4-12
- *--ignore_section=option[,option,...]* on page 4-44
- *--relax_symbol=option[,option,...]* on page 4-61.

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

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

4.51.1 Restrictions

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

4.51.2 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.

4.51.3 See also

Reference

- *--compare=option[,option,...]* on page 4-12
- *--ignore_symbol=option[,option,...]* on page 4-45
- *--relax_section=option[,option,...]* on page 4-60.

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

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

4.52.1 Restrictions

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

4.52.2 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
- specify multiple options in one `--rename` option followed by a comma-separated list of arguments.

4.52.3 Example

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

```
fromelf --cpu=8-A.32 --elf --rename=clock=myclock --output=mytimer.axf timer.axf
```

4.52.4 See also

Reference

- *--elf* on page 4-28
- *--output=destination* on page 4-57.

4.53 --select=*select_options*

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

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

4.53.1 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 specify multiple fields, 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.

4.53.2 See also

Reference

- *--fieldoffsets* on page 4-34
- *--text* on page 4-71.

4.54 --show=*option*[,*option*,...]

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

4.54.1 Restrictions

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

4.54.2 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.

4.54.3 See also

Reference

- *--elf* on page 4-28
- *--hide=option[,option,...]* on page 4-40
- *--output=destination* on page 4-57.

4.55 --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.

4.55.1 Restrictions

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

4.55.2 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.

4.55.3 See also

Reference

- *--elf* on page 4-28
- *--output=destination* on page 4-57.

4.56 --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.

4.56.1 See also

Reference

- *--via=filename* on page 4-75
- Chapter 4 *fromelf command reference*.

4.57 --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.

You can use this option with `--interleave`.

4.57.1 See also

Reference

- *--interleave=option* on page 4-50.

4.58 --strip=*option*[,*option*,...]

This option helps to protect your code in images and objects that are delivered to third parties. You can also use it to help reduce the size of the output image.

4.58.1 Restrictions

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

You must also use --cpu with this option in this release.

4.58.2 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 option 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.

  Note

  This option does not strip path names that are in the debug information.

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, path names, or file symbols might make the file harder to debug.

4.58.3 Example

To produce an `output.axf` file without debug from the ELF file `infile.axf` originally produced with debug, enter:

```
fromelf --cpu=8-A.32 --strip=debug,symbols --elf --output=outfile.axf infile.axf
```

4.58.4 See also

Concepts

Using the Linker:

- *About mapping symbols* on page 7-3.

Reference

- *--elf* on page 4-28
- *--linkview, --no_linkview* on page 4-52
- *--privacy* on page 4-58.

armlink Reference Guide:

- *--locals, --no_locals* on page 2-84
- *--privacy* on page 2-101.

4.59 --symbolversions, --no_symbolversions

This option turns off the decoding of symbol version tables.

4.59.1 Restrictions

If you use --elf with this option, you must also use --output.

4.59.2 See also

Reference

armlink Reference Guide:

- *About symbol versioning* on page 9-18.

Other information

- *Base Platform ABI for the ARM Architecture* http://infocenter.arm.com/help/topic/com.arm.doc.ihi0037-/index.html.

.

4.60 --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`.

4.60.1 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.

Note

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

4.60.2 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, enter:

  ```
  fromelf --cpu=8-A.32 --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, enter:

  ```
  fromelf --cpu=8-A.32 -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, enter:

  ```
  fromelf --cpu=8-A.32 --text -a --select=*.* --output=structaddress.txt infile.axf
  ```

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

  ```
  fromelf --cpu=8-A.32 --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, enter:

  ```
  fromelf --cpu=8-A.32 --text -a --select=*,~*.* --output=structaddress.txt
  infile.axf
  ```

4.60.3 See also

Tasks

- *Using fromelf to find where a symbol is placed in an executable ELF image* on page 3-13.

armlink User Guide:

- *Linker options for getting information about images* on page 6-2.

Reference

- *--cpu=name* on page 4-16
- *--disassemble* on page 4-26
- *--emit=option[,option,...]* on page 4-29
- *--expandarrays* on page 4-31
- *--info=topic[,topic,...]* on page 4-47
- *--interleave=option* on page 4-50
- *--only=section_name* on page 4-56
- *--output=destination* on page 4-57
- *--select=select_options* on page 4-63
- *-w* on page 4-77
- *--wide64bit* on page 4-78.

4.61 --version_number

This option displays the version of fromelf you are using.

4.61.1 Syntax

```
fromelf --version_number
```

fromelf displays the version number in the format nnbbbb, where:

- nn is the version number
- bbbb is the build number.

4.61.2 Example

Version 6.0 build 0019 is displayed as 600019.

4.61.3 See also

Reference

- *--help* on page 4-39
- *--vsn* on page 4-76.

4.62 --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.

4.62.1 Restrictions

You cannot use this option with object files.

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

4.62.2 See also

Concepts

- *Considerations when using fromelf* on page 2-4.

Reference

- *--output=destination* on page 4-57
- *--widthxbanks* on page 4-79.

4.63 --via=*filename*

Reads an additional list of input filenames and ELF file converter options from *filename*.

4.63.1 Syntax

```
--via=filename
```

Where filename is the name of a via file containing options to be included on the command line.

4.63.2 Usage

You can enter multiple `--via` options on the ELF file converter command line. The `--via` options can also be included within a via file.

4.63.3 See also

Reference

- Appendix A *Via File Syntax*.

4.64 --vsn

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

```
>fromelf --vsn
ARM FromELF, N.n [Build num] license_type
Software supplied by: ARM Limited
```

4.64.1 See also

Reference

- *--help* on page 4-39
- *--version_number* on page 4-73.

4.65 -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 --cpu=8-A.32 --text -w -c test.axf
================================================================================

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

** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]    Size   : 36 bytes (alignment 4)    Address:
0x00000000    $a
     .text
.
.
.
** Section #7 '.rel.text' (SHT_REL)    Size   : 8 bytes (alignment 4)    Symbol table #6 '.symtab'    1
relocations applied to section #1 '.text'
** Section #2 '.ARM.exidx' (SHT_ARM_EXIDX) [SHF_ALLOC + SHF_LINK_ORDER]    Size   : 8 bytes (alignment 4)
Address: 0x
00000000    Link to section #1 '.text'
** Section #8 '.rel.ARM.exidx' (SHT_REL)    Size   : 8 bytes (alignment 4)    Symbol table #6 '.symtab'    1
relocations applied to section #2 '.ARM.exidx'
** Section #3 '.arm_vfe_header' (SHT_PROGBITS)    Size   : 4 bytes (alignment 4)
** Section #4 '.comment' (SHT_PROGBITS)    Size   : 74 bytes
** Section #5 '.debug_frame' (SHT_PROGBITS)    Size   : 140 bytes
** Section #9 '.rel.debug_frame' (SHT_REL)    Size   : 32 bytes (alignment 4)    Symbol table #6 '.symtab'    4
relocations applied to section #5 '.debug_frame'
** Section #6 '.symtab' (SHT_SYMTAB)    Size   : 176 bytes (alignment 4)    String table #11 '.strtab'    Last
local symbol no. 5
** Section #10 '.shstrtab' (SHT_STRTAB)    Size   : 110 bytes
** Section #11 '.strtab' (SHT_STRTAB)    Size   : 223 bytes
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)    Size   : 69 bytes
```

4.65.1 See also

Reference

- *--text* on page 4-71.

4.66 --wide64bit

This option causes all addresses to be displayed with a width of 64 bits.

4.66.1 Usage

Without this option fromelf displays addresses as 32 bits where possible, and only displays them as 64 bits when necessary.

This option is ignored if the input file is not an AArch64 state file.

4.66.2 See also

Reference

- *input_file* on page 4-48.

4.67 --*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.

4.67.1 Restrictions

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

4.67.2 Syntax

--widthxbanks

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
```

4.67.3 Usage

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 --cpu=8-A.32 --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 --cpu=8-A.32 --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 --cpu=8-A.32 --vhx --8x4 test.axf --output=file` produces four files (`file0`, `file1`, `file2`, and `file3`). Each file contains lines of single bytes, for example:

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

- `fromelf --cpu=8-A.32 --vhx --16x2 test.axf --output=file` produces two files (`file0` and `file1`). Each file contains lines of two bytes, for example:

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

4.67.4 See also

Reference

- *--bin* on page 4-5
- *--output=destination* on page 4-57
- *--vhx* on page 4-74.

Appendix A
Via File Syntax

This appendix describes the syntax of via files accepted by all the ARM development tools:

A.1 Overview of via files

Via files are plain text files that contain command-line arguments and options to ARM development tools.

Typically, you can use a via file to overcome the command-line length limitations. However, you might want to create multiple via files that:

- Group similar arguments and options together.
- Contain different sets of arguments and options to be used in different scenarios.

Note

In general, you can use a via file to specify any command-line option to a tool, including `--via`. This means that you can call multiple nested via files from within a via file.

A.1.1 Via file evaluation

When fromelf is invoked it:

1. Replaces the first specified `--via` *`via_file`* argument with the sequence of argument words extracted from the via file, including recursively processing any nested `--via` commands in the via file.
2. Processes any subsequent `--via` *`via_file`* arguments in the same way, in the order they are presented.

That is, via files are processed in the order you specify them, and each via file is processed completely including processing nested via files before processing the next via file.

A.2 Via file syntax

Via files must conform to the following syntax rules:

- A via file is a text file containing a sequence of words. Each word in the text file is converted into an argument string and passed to the tool.
- Words are separated by whitespace, or the end of a line, except in delimited strings. For example:

  `--debugonly --privacy` (two words)

  `--debugonly--privacy` (one word)
- The end of a line is treated as whitespace. For example:

  ```
  --debugonly
  --privacy
  ```

  is equivalent to:

  ```
  --debugonly --privacy
  ```
- Strings enclosed in quotation marks ("), or apostrophes (') are treated as a single word. Within a quoted word, an apostrophe is treated as an ordinary character. Within an apostrophe delimited word, a quotation mark is treated as an ordinary character.

  Use quotation marks to delimit filenames or path names that contain spaces. For example:

  `--output C:\My Project\output.txt` (three words)

  `--output "C:\My Project\output.txt"` (two words)

  Use apostrophes to delimit words that contain quotes.
- Characters enclosed in parentheses are treated as a single word. For example:

  `--option(x, y, z)` (one word)

  `--option (x, y, z)` (two words)
- Within quoted or apostrophe delimited strings, you can use a backslash (\) character to escape the quote, apostrophe, and backslash characters.
- A word that occurs immediately next to a delimited word is treated as a single word. For example:

  `--output"C:\Project\output.txt"`

  is treated as the single word:

  `--outputC:\Project\output.txt`
- Lines beginning with a semicolon (;) or a hash (#) character as the first nonwhitespace character are comment lines. If a semicolon or hash character appears anywhere else in a line, it is not treated as the start of a comment. For example:

  ```
  -o objectname.axf      ;this is not a comment
  ```

  A comment ends at the end of a line, or at the end of the file. There are no multi-line comments, and there are no part-line comments.