## Document History

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>14 March 2014</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.00 Release</td>
</tr>
<tr>
<td>B</td>
<td>15 December 2014</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.01 Release</td>
</tr>
<tr>
<td>C</td>
<td>30 June 2015</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.02 Release</td>
</tr>
<tr>
<td>D</td>
<td>18 November 2015</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.3 Release</td>
</tr>
<tr>
<td>E</td>
<td>24 February 2016</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.4 Release</td>
</tr>
<tr>
<td>F</td>
<td>29 June 2016</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.5 Release</td>
</tr>
<tr>
<td>G</td>
<td>04 November 2016</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.6 Release</td>
</tr>
<tr>
<td>H</td>
<td>08 May 2017</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.6.1 Release</td>
</tr>
<tr>
<td>I</td>
<td>29 November 2017</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.6.2 Release</td>
</tr>
<tr>
<td>J</td>
<td>28 August 2019</td>
<td>Non-Confidential</td>
<td>Arm Compiler v6.6.3 Release</td>
</tr>
</tbody>
</table>

**Non-Confidential Proprietary Notice**

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is
not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at http://www.arm.com/company/policies/trademarks.

Copyright © 2014–2017, 2019 Arm Limited (or its affiliates). All rights reserved.


110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

www.arm.com
Contents

Arm® Compiler fromelf User Guide

Preface
About this book ........................................................................................................ 10

Chapter 1  Overview of the fromelf Image Converter
1.1 About the fromelf image converter ................................................................. 1-13
1.2 fromelf execution modes ................................................................................ 1-14
1.3 Getting help on the fromelf command ......................................................... 1-15
1.4 fromelf command-line syntax ....................................................................... 1-16
1.5 Support level definitions ................................................................................ 1-17

Chapter 2  Using fromelf
2.1 General considerations when using fromelf .................................................... 2-22
2.2 Examples of processing ELF files in an archive ............................................. 2-23
2.3 Options to protect code in image files with fromelf ..................................... 2-24
2.4 Options to protect code in object files with fromelf ..................................... 2-25
2.5 Option to print specific details of ELF files .................................................. 2-27
2.6 Using fromelf to find where a symbol is placed in an executable ELF image .... 2-28

Chapter 3  fromelf Command-line Options
3.1 --base [object_file::]load_region_ID=num ............................................... 3-32
3.2 --bin ........................................................................................................ 3-34
3.3 --bincombined ......................................................................................... 3-35
3.4 --bincombined_base=address .................................................................... 3-36
3.5 --bincombined_padding=size,num .......................................................... 3-37
--cad ................................................................. 3-38
--cadcombined .................................................. 3-40
--compare=option[,option,...] ............................... 3-41
--continue_on_error ............................................ 3-43
--cpu=list ........................................................... 3-44
--cpu=name .......................................................... 3-45
--datatypes ........................................................ 3-47
--debugonly .......................................................... 3-48
--decode_build_attributes ....................................... 3-49
--diag_error=tag[,tag,...] ...................................... 3-51
--diag_remark=tag[,tag,...] ..................................... 3-52
--diag_style={arm|ide|gnu} .................................... 3-53
--diag_suppress=tag[,tag,...] .................................. 3-54
--diag_warning=tag[,tag,...] .................................... 3-55
--disassemble ...................................................... 3-56
--dump_build_attributes ......................................... 3-57
--elf ................................................................. 3-58
--emit=option[,option,...] ..................................... 3-59
--expandarrays .................................................... 3-61
--extract_build_attributes ..................................... 3-62
--fieldoffsets ..................................................... 3-63
--fpulist .............................................................. 3-65
--fpu=name .......................................................... 3-66
--globalize=option[,option,...] ............................. 3-67
--help .................................................................. 3-68
--hide=option[,option,...] ...................................... 3-69
--hide_and_localize=option[,option,...] ................. 3-70
--i32 ................................................................... 3-71
--i32combined ..................................................... 3-72
--ignore_section=option[,option,...] ....................... 3-73
--ignore_symbol=option[,option,...] ....................... 3-74
--in_place .......................................................... 3-75
--info=topic[,topic,...] ........................................ 3-76
--input_file ........................................................ 3-77
--interleave=option .............................................. 3-79
--linkview, --no_linkview ..................................... 3-80
--localize=option[,option,...] ............................... 3-81
--m32combined .................................................... 3-82
--m32 ................................................................. 3-82
--m32combined .................................................... 3-83
--only=section_name ............................................. 3-84
--output=destination ............................................ 3-85
--privacy ............................................................ 3-86
--qualify ............................................................. 3-87
--relax_section=option[,option,...] ......................... 3-88
--relax_symbol=option[,option,...] ......................... 3-89
--rename=option[,option,...] .................................. 3-90
--select=select_options ....................................... 3-91
--show=option[,option,...] .................................... 3-92
--show_and_globalize=option[,option,...] ............... 3-93
--show_cmdline ................................................... 3-94
Chapter 4

Via File Syntax

4.1 Overview of via files .......................................................... 4-110
4.2 Via file syntax rules .......................................................... 4-111
List of Figures

Arm® Compiler fromelf User Guide

Figure 1-1 Integration boundaries in Arm Compiler 6. ................................................................. 1-19
List of Tables
Arm® Compiler fromelf User Guide

Table 2-1  Effect of fromelf --privacy and --strip options on images files ................................................ 2-24
Table 2-2  Effect of fromelf --privacy and --strip options on object files .................................................. 2-25
Table 3-1  Examples of using --base ...................................................................................................... 3-32
Table 3-2  Supported Arm architectures ................................................................................................. 3-45
Preface

This preface introduces the *Arm® Compiler fromelf User Guide*. It contains the following:

- *About this book on page 10.*
About this book

Arm® Compiler fromelf User Guide provides information on how to use the fromelf utility.

Using this book

This book is organized into the following chapters:

*Chapter 1 Overview of the fromelf Image Converter*
  Gives an overview of the fromelf image converter provided with Arm Compiler.

*Chapter 2 Using fromelf*
  Describes how to use the fromelf image converter provided with Arm Compiler.

*Chapter 3 fromelf Command-line Options*
  Describes the command-line options of the fromelf image converter provided with Arm Compiler.

*Chapter 4 Via File Syntax*
  Describes the syntax of via files accepted by the armasm, armlink, fromelf, and armar tools.

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm® Glossary* for more information.

Typographic conventions

*italic*

Introduces special terminology, denotes cross-references, and citations.

*bold*

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

*monospace*

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

*monospace*

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

*monospace italic*

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

*monospace bold*

Denotes language keywords when used outside example code.

*<and>*

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

**SMALL CAPITALS**

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Feedback

Feedback on this product
If you have any comments or suggestions about this product, contact your supplier and give:
• The product name.
• The product revision or version.
• An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content
If you have comments on content then send an e-mail to errata@arm.com. Give:
• The title Arm Compiler fromelf User Guide.
• The number DUI0805J.
• If applicable, the page number(s) to which your comments refer.
• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note
Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information
• Arm® Developer.
• Arm® Information Center.
• Arm® Technical Support Knowledge Articles.
• Technical Support.
• Arm® Glossary.
Chapter 1
Overview of the fromelf Image Converter

Gives an overview of the fromelf image converter provided with Arm Compiler.

It contains the following sections:

• 1.1 About the fromelf image converter on page 1-13.
• 1.2 fromelf execution modes on page 1-14.
• 1.3 Getting help on the fromelf command on page 1-15.
• 1.4 fromelf command-line syntax on page 1-16.
• 1.5 Support level definitions on page 1-17.
1.1 About the fromelf image converter

The fromelf image conversion utility allows you to modify ELF image and object files, and to display information on those files.

fromelf allows you to:
- Process Arm ELF object and image files that the compiler, assembler, and linker generate.
- Process all ELF files in an archive that armar creates, and output the processed files into another archive if necessary.
- Convert ELF images into other formats for use by ROM tools or for direct loading 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.
- Display information about the input file, for example, disassembly output or symbol listings, to either stdout or a text file. Disassembly is generated in armasm assembler syntax and not GNU assembler syntax.

---------- Note ----------
If your image is produced without debug information, fromelf cannot:
- Translate the image into other file formats.
- Produce a meaningful disassembly listing.

---------- Note ----------
The command-line option descriptions and related information in the individual Arm Compiler tools documents describe all the features that Arm Compiler supports. Any features not documented are not supported and are used at your own risk. You are responsible for making sure that any generated code using community features on page 1-17 is operating correctly.

Related concepts
2.3 Options to protect code in image files with fromelf on page 2-24
2.4 Options to protect code in object files with fromelf on page 2-25

Related references
1.2 fromelf execution modes on page 1-14
1.4 fromelf command-line syntax on page 1-16
Chapter 3 fromelf Command-line Options on page 3-30
1.2 fromelf execution modes

You can run fromelf in various execution modes.

The execution modes are:

• 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).

Related references

3.2 --bin on page 3-34
3.22 --elf on page 3-58
3.33 --i32 on page 3-71
3.43 --m32 on page 3-82
3.59 --text on page 3-99
3.61 --vhx on page 3-102
1.3 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.

To display the help information, enter:

```
fromelf --help
```

Related references
1.4 fromelf command-line syntax on page 1-16
3.30 --help on page 3-68
1.4 **fromelf command-line syntax**

You can specify an ELF file or library of ELF files on the `fromelf` command-line.

**Syntax**

```bash
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.

**Related references**

*Chapter 3 fromelf Command-line Options on page 3-30*

*3.39 input_file on page 3-77*
1.5 Support level definitions

This describes the levels of support for various Arm Compiler 6 features.

Arm Compiler 6 is built on Clang and LLVM technology. Therefore it has more functionality than the set of product features described in the documentation. The following definitions clarify the levels of support and guarantees on functionality that are expected from these features.

Arm welcomes feedback regarding the use of all Arm Compiler 6 features, and endeavors to support users to a level that is appropriate for that feature. You can contact support at https://developer.arm.com/support.

Identification in the documentation

All features that are documented in the Arm Compiler 6 documentation are product features, except where explicitly stated. The limitations of non-product features are explicitly stated.

Product features

Product features are suitable for use in a production environment. The functionality is well-tested, and is expected to be stable across feature and update releases.

- Arm endeavors to give advance notice of significant functionality changes to product features.
- If you have a support and maintenance contract, Arm provides full support for use of all product features.
- Arm welcomes feedback on product features.
- Any issues with product features that Arm encounters or is made aware of are considered for fixing in future versions of Arm Compiler.

In addition to fully supported product features, some product features are only alpha or beta quality.

Beta product features

Beta product features are implementation complete, but have not been sufficiently tested to be regarded as suitable for use in production environments.

Beta product features are indicated with [BETA].

- Arm endeavors to document known limitations on beta product features.
- Beta product features are expected to eventually become product features in a future release of Arm Compiler 6.
- Arm encourages the use of beta product features, and welcomes feedback on them.
- Any issues with beta product features that Arm encounters or is made aware of are considered for fixing in future versions of Arm Compiler.

Alpha product features

Alpha product features are not implementation complete, and are subject to change in future releases, therefore the stability level is lower than in beta product features.

Alpha product features are indicated with [ALPHA].

- Arm endeavors to document known limitations of alpha product features.
- Arm encourages the use of alpha product features, and welcomes feedback on them.
- Any issues with alpha product features that Arm encounters or is made aware of are considered for fixing in future versions of Arm Compiler.

Community features

Arm Compiler 6 is built on LLVM technology and preserves the functionality of that technology where possible. This means that there are additional features available in Arm Compiler that are not listed in the documentation. These additional features are known as community features. For information on these community features, see the documentation for the Clang/LLVM project.
Where community features are referenced in the documentation, they are indicated with [COMMUNITY].

- Arm makes no claims about the quality level or the degree of functionality of these features, except when explicitly stated in this documentation.
- Functionality might change significantly between feature releases.
- Arm makes no guarantees that community features will remain functional across update releases, although changes are expected to be unlikely.

Some community features might become product features in the future, but Arm provides no roadmap for this. Arm is interested in understanding your use of these features, and welcomes feedback on them. Arm supports customers using these features on a best-effort basis, unless the features are unsupported. Arm accepts defect reports on these features, but does not guarantee that these issues will be fixed in future releases.

**Guidance on use of community features**

There are several factors to consider when assessing the likelihood of a community feature being functional:

- The following figure shows the structure of the Arm Compiler 6 toolchain:
Figure 1-1 Integration boundaries in Arm Compiler 6.

The dashed boxes are toolchain components, and any interaction between these components is an integration boundary. Community features that span an integration boundary might have significant limitations in functionality. The exception to this is if the interaction is codified in one of the standards supported by Arm Compiler 6. See Application Binary Interface (ABI) for the Arm® Architecture. Community features that do not span integration boundaries are more likely to work as expected.

- Features primarily used when targeting hosted environments such as Linux or BSD might have significant limitations, or might not be applicable, when targeting bare-metal environments.
- The Clang implementations of compiler features, particularly those that have been present for a long time in other toolchains, are likely to be mature. The functionality of new features, such as support
for new language features, is likely to be less mature and therefore more likely to have limited functionality.

**Unsupported features**

With both the product and community feature categories, specific features and use-cases are known not to function correctly, or are not intended for use with Arm Compiler 6.

Limitations of product features are stated in the documentation. Arm cannot provide an exhaustive list of unsupported features or use-cases for community features. The known limitations on community features are listed in *Community features on page 1-17.*

**List of known unsupported features**

The following is an incomplete list of unsupported features, and might change over time:

- The Clang option `-stdlib=libstdc++` is not supported.
- C++ static initialization of local variables is not thread-safe when linked against the standard C++ libraries. For thread-safety, you must provide your own implementation of thread-safe functions as described in *Standard C++ library implementation definition.*

  **Note**

  This restriction does not apply to the [ALPHA]-supported multi-threaded C++ libraries.

- Use of C11 library features is unsupported.
- Any community feature that exclusively pertains to non-Arm architectures is not supported.
- Compilation for targets that implement architectures older than Armv7 or Armv6-M is not supported.
- The `long double` data type is not supported for AArch64 state because of limitations in the current Arm C library.
- Complex numbers are not supported because of limitations in the current Arm C library.
Chapter 2
Using fromelf

Describes how to use the fromelf image converter provided with Arm Compiler.

It contains the following sections:
• 2.1 General considerations when using fromelf on page 2-22.
• 2.2 Examples of processing ELF files in an archive on page 2-23.
• 2.3 Options to protect code in image files with fromelf on page 2-24.
• 2.4 Options to protect code in object files with fromelf on page 2-25.
• 2.5 Option to print specific details of ELF files on page 2-27.
• 2.6 Using fromelf to find where a symbol is placed in an executable ELF image on page 2-28.
2.1 General considerations when using fromelf

There are some changes that you cannot make to an image with fromelf.

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.

Related references

3.1 --base [object_file::load_region_ID=num on page 3-32
3.39 input_file on page 3-77
2.2 Examples of processing ELF files in an archive

Examples of how 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.

Examples

Consider an archive, test.a, containing the following ELF files:

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

Example of processing all files in the archive

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

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

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

Example of processing a subset of files in the archive

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

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

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

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

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

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

Related references

3.20 --disassemble on page 3-56
3.22 --elf on page 3-58
3.39 input_file on page 3-77
3.46 --output=destination on page 3-85
3.57 --strip=option[,option,...] on page 3-96
2.3 Options to protect code in image files with fromelf

If you are delivering images to third parties, then you might want to protect the code they contain.

To help you to protect this code, fromelf provides the \texttt{--strip} option and the \texttt{--privacy} option. These options remove or obscure the symbol names in the image. The option that you choose depends on how much information you want to remove. The effect of these options is different for image files.

\textbf{Restrictions}

You must use \texttt{--elf} with these options. Because you have to use \texttt{--elf}, you must also use \texttt{--output}.

\textbf{Effect of the options for protecting code in image files}

For image files:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Option & Effect \\
\hline
\texttt{fromelf --elf --privacy} & Removes the whole symbol table. \\
& Removes the \texttt{.comment} section name. This section is marked as [Anonymous Section] in the \texttt{fromelf --text} output. \\
& Gives section names a default value. For example, changes code section names to '.text'. \\
\hline
\texttt{fromelf --elf --strip=symbols} & Removes the whole symbol table. \\
& Section names remain the same. \\
\hline
\texttt{fromelf --elf --strip=localsymbols} & Removes local and mapping symbols. \\
& Retains section names and build attributes. \\
\hline
\end{tabular}
\caption{Effect of fromelf \texttt{--privacy} and \texttt{--strip} options on images files}
\end{table}

\textbf{Example}

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

\begin{verbatim}
fromelf --elf --privacy --output=outfile.axf infile.axf
\end{verbatim}

\textbf{Related concepts}

2.4 Options to protect code in object files with fromelf on page 2-25

\textbf{Related references}

1.4 fromelf command-line syntax on page 1-16
3.22 \texttt{--elf} on page 3-58
3.46 \texttt{--output=destination} on page 3-85
3.47 \texttt{--privacy} on page 3-86
3.57 \texttt{--strip=option[,option,...]} on page 3-96
2.4 Options to protect code in object files with fromelf

If you are delivering 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. The option you choose depends on how much information you want to remove. The effect of these options is different for object files.

Restrictions

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

Effect of the options for protecting code in object files

For object files:

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

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

Example

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

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

Related concepts

2.3 Options to protect code in image files with fromelf on page 2-24

Related references

1.4 fromelf command-line syntax on page 1-16
3.22 --elf on page 3-58
3.46 --output=destination on page 3-85
3.47 --privacy on page 3-86
3.57 --strip=option[,option,...] on page 3-96
2.5 Option to print specific details of ELF 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.

Examples

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

```
fromelf --emit=data infile.axf
```

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

```
fromelf --emit=relocation_tables,dynamic_segment infile2.axf
```

Related references

1.4 fromelf command-line syntax on page 1-16
3.23 `--emit=option[,option,...]` on page 3-59
3.59 `--text` on page 3-99
2.6 Using fromelf to find where a symbol is placed in an executable ELF image

You can 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:

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

Procedure

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

   ```c
   #include <stdio.h>
   #include <stdlib.h>

   long long arr[10] __attribute__((section("ARRAY")));
   int main()
   {
       return sizeof(arr);
   }
   ```

2. Compile the source:

   ```bash
   armclang --target=arm-arm-none-eabi -march=armv8-a -c s.c -o s.o
   ```

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

   ```bash
   armlink --cpu=8-A.32 --keep=s.o(ARRAY) s.o --output=s.axf
   ```

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

   ```bash
   fromelf --text -s -v s.o
   ```

5. Locate the `arr` symbol in the fromelf output, for example:

   ```bash
   ** Section #24
   Name : .symtab
   Type : SHT_SYMTAB (0x00000002)
   Flags : None (0x00000000)
   Addr : 0x00000000
   File Offset : 868 (0x364)
   Size : 464 bytes (0x1d0)
   Link : Section 1 (.strtab)
   Info : Last local symbol no = 26
   Alignment : 4
   Entry Size : 16
   Symbol table .symtab (28 symbols, 26 local)
   # Symbol Name                Value      Bind  Sec  Type  Vis  Size
   =========================================================================
   ... 27  arr                        0x00000000   Gb    5  Data  De   0x50
   ...```

   The `Sec` column shows the section where the stack is placed. In this example, section 5.

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

   ```bash
   ** Section #5
   Name : ARRAY
   Type : SHT_PROGBITS (0x00000001)
   Flags : SHF_ALLOC + SHF_WRITE (0x00000003)
   Addr : 0x00000000
   File Offset : 88 (0x58)
   Size : 80 bytes (0x50)
   Link : SHN_UNDEF
   Info : 0
   Alignment : 8
   Entry Size : 0
   ```

   This shows that the symbols are placed in an ARRAY section.
Related references

3.59 --text on page 3-99
Chapter 3
fromelf Command-line Options

Describes the command-line options of the fromelf image converter provided with Arm Compiler.

It contains the following sections:

- 3.1 --base [object_file::]load_region_ID=num on page 3-32.
- 3.2 --bin on page 3-34.
- 3.3 --bincombined on page 3-35.
- 3.4 --bincombined_base=address on page 3-36.
- 3.5 --bincombined_padding=size,num on page 3-37.
- 3.6 --cad on page 3-38.
- 3.7 --cadcombined on page 3-40.
- 3.8 --compare=option[,option,...] on page 3-41.
- 3.9 --continue_on_error on page 3-43.
- 3.10 --cpu=list on page 3-44.
- 3.11 --cpu=name on page 3-45.
- 3.12 --datasyncs on page 3-47.
- 3.14 --decode_build_attributes on page 3-49.
- 3.15 --diag_error=tag[,tag,...] on page 3-51.
- 3.16 --diag_remark=tag[,tag,...] on page 3-52.
- 3.17 --diag_style={arm|ide|gnu} on page 3-53.
- 3.18 --diag_suppress=tag[,tag,...] on page 3-54.
- 3.19 --diag_warning=tag[,tag,...] on page 3-55.
- 3.20 --disassemble on page 3-56.
- 3.21 --dump_build_attributes on page 3-57.
- 3.22 --elf on page 3-58.
- 3.23 --emit=option[,option,...] on page 3-59.
3.24 --expandarrays on page 3-61.
3.25 --extract_build_attributes on page 3-62.
3.26 --fieldoffsets on page 3-63.
3.27 --fpu=list on page 3-65.
3.28 --fpu=name on page 3-66.
3.29 --globalize=option[,option,...] on page 3-67.
3.30 --help on page 3-68.
3.31 --hide=option[,option,...] on page 3-69.
3.32 --hide_and_localize=option[,option,...] on page 3-70.
3.33 --i32 on page 3-71.
3.34 --i32combined on page 3-72.
3.35 --ignore_section=option[,option,...] on page 3-73.
3.36 --ignore_symbol=option[,option,...] on page 3-74.
3.37 --in_place on page 3-75.
3.38 --info=topic[,topic,...] on page 3-76.
3.39 input_file on page 3-77.
3.40 --interleave=option on page 3-79.
3.41 --linkview, --no_linkview on page 3-80.
3.42 --localize=option[,option,...] on page 3-81.
3.43 --m32 on page 3-82.
3.44 --m32combined on page 3-83.
3.45 --only=section_name on page 3-84.
3.46 --output=destination on page 3-85.
3.47 --privacy on page 3-86.
3.48 --qualify on page 3-87.
3.49 --relax_section=option[,option,...] on page 3-88.
3.50 --relax_symbol=option[,option,...] on page 3-89.
3.51 --rename=option[,option,...] on page 3-90.
3.52 --select=select_options on page 3-91.
3.53 --show=option[,option,...] on page 3-92.
3.54 --show_and_globalize=option[,option,...] on page 3-93.
3.55 --show_cmdline on page 3-94.
3.56 --source_directory=path on page 3-95.
3.57 --strip=option[,option,...] on page 3-96.
3.58 --symbolversions, --no_symbolversions on page 3-98.
3.59 --text on page 3-99.
3.60 --version_number on page 3-101.
3.61 --vhx on page 3-102.
3.62 --via=file on page 3-103.
3.63 --vsn on page 3-104.
3.64 -w on page 3-105.
3.65 --wide64bit on page 3-106.
3.66 --widthxbanks on page 3-107.


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

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

--- Note ---

Not supported for AArch64 state inputs.

---

**Syntax**

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

Where:

- **object_file**
  An optional ELF input file.

- **load_region_ID**
  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**
  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 values in one `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.

**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.

**Examples**

The following table shows examples:

<table>
<thead>
<tr>
<th><code>--base</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0</code></td>
<td>decimal value</td>
</tr>
<tr>
<td><code>0x8000</code></td>
<td>hexadecimal value</td>
</tr>
<tr>
<td><code>#0=0</code></td>
<td>base address for the first load region</td>
</tr>
<tr>
<td><code>foo.o::*=0</code></td>
<td>base address for all load regions in <code>foo.o</code></td>
</tr>
<tr>
<td><code>#0=0,#1=0x8000</code></td>
<td>base address for the first and second load regions</td>
</tr>
</tbody>
</table>

**Related concepts**

2.1 General considerations when using fromelf on page 2-22

**Related references**

3.33 --i32 on page 3-71
3.34 --i32combined on page 3-72
3.43 --m32 on page 3-82
3.44 --m32combined on page 3-83
3.2 --bin

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.

Restrictions
The following restrictions apply:
• You cannot use this option with object files.
• You must use --output with this option.

Considerations when using --bin
If you convert an ELF image containing multiple load regions to a binary format, 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 only created when the load region describes code or data that is present in the ELF file. For example a load region containing only execution regions with ZI data in them does not result in an output file.

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

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

Related references
3.46 --output=destination on page 3-85
3.66 --widthxbanks on page 3-107
3.3 --bincombined

Produces plain binary output. It generates one output file for an image containing multiple load regions.

Usage

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.

Restrictions

The following restrictions apply:

- You cannot use this option with object files.
- You must use --output with this option.

Considerations when using --bincombined

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 0x2FFFFFF00 bytes.

Arm recommends that you use a different method of placing widely spaced load regions, such as --bin, and make your own arrangements to load the multiple output files at the correct addresses.

Examples

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

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

Related references

3.4 --bincombined_base=address on page 3-36
3.5 --bincombined_padding=size,num on page 3-37
3.46 --output=destination on page 3-85
3.66 --widthxbanks on page 3-107

Related information

Input sections, output sections, regions, and Program Segments
3.4 **--bincombined\_base=address**

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.

**Syntax**

```
--bincombined_base=address
```

Where *address* is the start address where the image is to be loaded:

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

**Default**

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

**Restrictions**

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

**Example**

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

**Related references**

- 3.3 **--bincombined** on page 3-35
- 3.5 **--bincombined_padding=size,num** on page 3-37

**Related information**

*Input sections, output sections, regions, and Program Segments*
3.5 \texttt{--bincombined\_padding=size,num}

Enables you to specify a different padding value from the default used by the \texttt{--bincombined} output mode.

\textbf{Syntax}

\texttt{--bincombined\_padding=size,num}

Where:

\textit{size} \\
Is 1, 2, or 4 bytes to define whether it is a byte, halfword, or word.

\textit{num} \\
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.

\underline{Note} from\texttt{elf} 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.

\textbf{Default}

The default is \texttt{--bincombined\_padding=1,0xFF}.

\textbf{Restrictions}

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

\textbf{Examples}

The following examples show how to use \texttt{--bincombined\_padding}:

\texttt{--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 \texttt{0x12345678}.

\texttt{--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 \texttt{0x1234}.

\texttt{--bincombined --bincombined\_padding=2,0x01}

This example when specified for big endian memory, fills the space between load regions with \texttt{0x0100}.

\textbf{Related references}

3.3 \texttt{--bincombined} on page 3-35
3.4 \texttt{--bincombined\_base=address} on page 3-36
3.6 --cad

Produces a C array definition or C++ array definition containing binary output.

Usage

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.

Restrictions

You cannot use this option with object files.

Considerations when using --cad

A file is only created when the load region describes code or data that is present in the ELF file. For example a load region containing only execution regions with ZI data in them does not result in an output file.

Example

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

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

  ```
  fromelf --cad myimage.axf
  unsigned char LR0[] = {
      0x00,0x00,0x00,0xEB,0x28,0x00,0x00,0xEB,0x2C,0x00,0x8F,0xE2,0x00,0x00,0x0C,0x90,0xE8,
      0x00,0x0A,0x8A,0xEB,0x00,0x00,0x08,0xE0,0x01,0x70,0x4A,0xE2,0x08,0x00,0x5A,0x1E,  
      0x00,0x00,0x00,0x1A,0x2B,0x00,0x00,0xEB,0x0F,0x00,0x8A,0xEB,0x18,0xEB,0x4F,0xE2,  
      0x01,0x00,0x13,0xE3,0x03,0xF0,0x47,0x10,0x83,0xF0,0xA0,0xE1,0xA0,0x18,0x00,0x00,  
      0x0B,0x18,0x00,0x00,0x00,0x30,0x80,0xBE,0x00,0x40,0x80,0xE3,0x00,0x50,0x80,0xBE,  
      0x00,0x00,0x00,0x3E,0x10,0x20,0x52,0xE2,0x78,0x80,0xA1,0x2B,0xFC,0xFF,0x8A,  
      0x82,0x2E,0x00,0xE1,0x30,0x0E,0xA1,0x28,0x00,0x30,0x81,0x45,0x8E,0x80,0xA0,0xE1,  
      0x70,0x00,0x51,0xE3,0x3E,0x18,0x00,0x00,0x04,0x80,0x0A,0x1E,0x80,0x00,0x00,0x03,  
      0x00,0x00,0x00,0x0E,0xF0,0xA0,0xE1,0x1F,0x40,0x2D,0xE9,0x00,0x00,0xA0,0xE1,  
      ...
      ...
      0x3A,0x74,0x74,0x00,0x43,0x6F,0x6E,0x73,0x72,0x75,0x63,0x74,0x65,0x64,0x20,  
      0x41,0x20,0x23,0x25,0x64,0x20,0x61,0x74,0x20,0x25,0x70,0x6A,0x00,0x00,0x00,  
      0x80,0x44,0x65,0x73,0x74,0x72,0x6F,0x79,0x65,0x64,0x20,0x41,0x20,0x23,0x25,0x64,  
      0x20,0x61,0x74,0x20,0x25,0x70,0x6A,0x00,0x00,0x00,0x8C,0x99,0x00,0x00,0x0C,  
      0x99,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
      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.

Related references

3.7 --cadcombined on page 3-40
3.46 --output=destination on page 3-85
Related information
Input sections, output sections, regions, and Program Segments
3.7 --cadcombined

Produces a C array definition or C++ array definition containing binary output.

Usage

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.

Restrictions

You cannot use this option with object files.

Example

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

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

Related references

- 3.6 --cad on page 3-38
- 3.46 --output=destination on page 3-85
3.8 \texttt{--compare=option[,option,...]} \\

Compares two input files and prints a textual list of the differences.

**Usage**

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 \texttt{--relax_section} option.

**Syntax**

\texttt{--compare=option[,option,...]} 

Where \textit{option} is one of:

\textbf{section_sizes}

- Compares the size of all sections for each ELF file or ELF member of a library file.
  
  \textbf{section_sizes:::object_name}
  
  Compares the sizes of all sections in ELF objects with a name matching \textit{object_name}.

- Compares the sizes of all sections with a name matching \textit{section_name}.

\textbf{sections}

- Compares the size and contents of all sections for each ELF file or ELF member of a library file.
  
  \textbf{sections:::object_name}
  
  Compares the size and contents of all sections in ELF objects with a name matching \textit{object_name}.

- Compares the size and contents of all sections with a name matching \textit{section_name}.

\textbf{function_sizes}

- Compares the size of all functions for each ELF file or ELF member of a library file.
  
  \textbf{function_sizes:::object_name}
  
  Compares the size of all functions in ELF objects with a name matching \textit{object_name}.

- Compares the size of all functions with a name matching \textit{function_name}.

\textbf{global_function_sizes}

- Compares the size of all global functions for each ELF file or ELF member of a library file.
  
  \textbf{global_function_sizes:::function_name}
  
  Compares the size of all global functions in ELF objects with a name matching \textit{function_name}.

You can:

- Use wildcard characters ? and * for symbolic names in \textit{section_name}, \textit{function_name}, and \textit{object_name} arguments.
- Specify multiple values in one \textit{option} followed by a comma-separated list of arguments.
Related references
3.35 --ignore_section=option[,option,...] on page 3-73
3.36 --ignore_symbol=option[,option,...] on page 3-74
3.49 --relax_section=option[,option,...] on page 3-88
3.50 --relax_symbol=option[,option,...] on page 3-89
3.9 --continue_on_error

Reports any errors and then continues.

Usage

Use --diag_warning=error instead of this option.

Related references

3.19 --diag_warning=tag[,tag,...] on page 3-55
3.10  --cpu=list

Lists the architecture and processor names that are supported by the --cpu=name option.

Syntax

--cpu=list

Related references

3.11 --cpu=name on page 3-45
3.11 --cpu=name

Affects the way machine code is disassembled by options such as -c or --disassemble, so that it is disassembled in the same way that the specified processor interprets it.

Syntax

--cpu=name

Where name is the name of a processor or architecture:

Processor and architecture names are not case-sensitive.

Wildcard characters are not accepted.

The following table shows the supported architectures. For a complete list of the supported architecture and processor names, specify the --cpu=list option.

Table 3-2 Supported Arm architectures

<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-M</td>
<td>Armv6 architecture microcontroller profile.</td>
</tr>
<tr>
<td>6S-M</td>
<td>Armv6 architecture microcontroller profile with OS extensions.</td>
</tr>
<tr>
<td>7-A</td>
<td>Armv7 architecture application profile.</td>
</tr>
<tr>
<td>7-A.security</td>
<td>Armv7-A architecture profile with Security Extensions and includes the SMC instruction (formerly SMI).</td>
</tr>
<tr>
<td>7-R</td>
<td>Armv7 architecture real-time profile.</td>
</tr>
<tr>
<td>7-M</td>
<td>Armv7 architecture microcontroller profile.</td>
</tr>
<tr>
<td>7E-M</td>
<td>Armv7-M architecture profile with DSP extension.</td>
</tr>
<tr>
<td>8-A.32</td>
<td>Armv8-A architecture profile, AArch32 state.</td>
</tr>
<tr>
<td>8-A.32.crypto</td>
<td>Armv8-A architecture profile, AArch32 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8-A.64</td>
<td>Armv8-A architecture profile, AArch64 state.</td>
</tr>
<tr>
<td>8-A.64.crypto</td>
<td>Armv8-A architecture profile, AArch64 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.1-A.32.crypto</td>
<td>Armv8.1, for Armv8-A architecture profile, AArch32 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.1-A.64</td>
<td>Armv8.1, for Armv8-A architecture profile, AArch64 state.</td>
</tr>
<tr>
<td>8.1-A.64.crypto</td>
<td>Armv8.1, for Armv8-A architecture profile, AArch64 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.2-A.32</td>
<td>Armv8.2, for Armv8-A architecture profile, AArch32 state.</td>
</tr>
<tr>
<td>8.2-A.32.crypto</td>
<td>Armv8.2, for Armv8-A architecture profile, AArch32 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.2-A.64</td>
<td>Armv8.2, for Armv8-A architecture profile, AArch64 state.</td>
</tr>
<tr>
<td>8.2-A.64.crypto</td>
<td>Armv8.2, for Armv8-A architecture profile, AArch64 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.3-A.32</td>
<td>Armv8.3, for Armv8-A architecture profile, AArch32 state.</td>
</tr>
<tr>
<td>8.3-A.32.crypto</td>
<td>Armv8.3, for Armv8-A architecture profile, AArch32 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8.3-A.64</td>
<td>Armv8.3, for Armv8-A architecture profile, AArch64 state.</td>
</tr>
</tbody>
</table>
Table 3-2 Supported Arm architectures (continued)

<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3-A.64.crypto</td>
<td>Armv8.3, for Armv8-A architecture profile, AArch64 state with cryptographic instructions.</td>
</tr>
<tr>
<td>8-R</td>
<td>Armv8-R architecture profile.</td>
</tr>
<tr>
<td>8-M.Base</td>
<td>Armv8-M baseline architecture profile. Derived from the Armv6-M architecture.</td>
</tr>
<tr>
<td>8-M.Main</td>
<td>Armv8-M mainline architecture profile. Derived from the Armv7-M architecture.</td>
</tr>
<tr>
<td>8-M.Main.dsp</td>
<td>Armv8-M mainline architecture profile with DSP extension.</td>
</tr>
</tbody>
</table>

Note

The full list of supported architectures and processors depends on your license.

Usage

The following general points apply to processor and architecture options:

Processors

- Selecting the processor selects the appropriate architecture, Floating-Point Unit (FPU), and memory organization.

Architectures

- If you specify an architecture name for the --cpu option, machine code is disassembled by options such as -c or --disassemble for that architecture. If you specify --disassemble, then the disassembly can be assembled for any processor supporting that architecture.

For example, --cpu=7-A --disassemble produces disassembly that can be assembled for the Cortex®-A7 processor.

FPU

- Some specifications of --cpu imply an --fpu selection.

Note

Any explicit FPU, set with --fpu on the command line, overrides an implicit FPU.

- If no --fpu option is specified and no --cpu option is specified, --fpu=softvfp is used.

Default

If you do not specify a --cpu option, then fromelf disassembles machine instructions in an architecture-independent way. This means that fromelf disassembles anything that it recognizes as an instruction by some architecture.

Example

To specify the Cortex-M4 processor, use:

```bash
--cpu=Cortex-M4
```

Related references

3.10 --cpu=list on page 3-44
3.20 --disassemble on page 3-56
3.38 --info=topic[,topic,...] on page 3-76
3.59 --text on page 3-99
3.12 --datasymbols

Modifies the output information of data sections so that symbol definitions are interleaved.

Usage

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

Related references

3.59 --text on page 3-99
3.13 --debugonly

Removes the content of any code or data sections.

Usage

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

Restrictions

You must use --elf with this option.

Example

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

```
fromelf --elf --debugonly --output=debugout.axf infile.axf
```

Related references

3.22 --elf on page 3-58
3.14 --decode_build_attributes

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

--- Note ---

The standard build attributes are documented in the Application Binary Interface for the Arm® Architecture.

**Restrictions**

This option has no effect for AArch64 state inputs.

**Example**

The following example shows the output for --decode_build_attributes:

```
armclang --target=arm-arm-eabi-none -march=armv8-a -c hello.c -o hello.o
fromelf -v --decode_build_attributes hello.o
```

```
** Section #6
Name     : .ARM.attributes
Type     : SHT_ARM_ATTRIBUTES (0x70000003)
Flags    : None (0x00000000)
Addr     : 0x00000000
File Offset : 112 (0x70)
Size     : 74 bytes (0x4a)
Link     : SHN_UNDEF
Info     : 0
Alignment : 1
Entry Size  : 0

'aebi' file build attributes:
0x000000:   43 32 2e 30 39 00 05 63 6f 72 74 65 78 2d 61 35  
0x000010:   33 00 06 0e 07 41 08 01 09 02 0a 07 0c 03 0e 00  
0x000020:   11 01 12 04 14 01 15 01 17 03 18 01 19 01 1a 02  
0x000030:   22 00 24 01 26 01 2a 01 44 03                      

Tag_conformance = "2.09"
Tag_CPU_name = "cortex-a53"
Tag_CPU_arch = ARM v8 (=14)
Tag_CPU_arch_profile = The application profile 'A' (e.g. for Cortex A8)
```

```
```

DUI0805J  Copyright © 2014–2017, 2019 Arm Limited or its affiliates. All rights reserved.
Non-Confidential
was permitted (=3)
...

Related references
3.21 --dump_build_attributes on page 3-57
3.23 --emit=option[,option,...] on page 3-59
3.25 --extract_build_attributes on page 3-62

Related information
Application Binary Interface for the ARM Architecture
3.15 --diag_error=tag[,tag,...]

Sets diagnostic messages that have a specific tag to Error severity.

Syntax

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

Where tag can be:

• A diagnostic message number to set to error severity. This is the four-digit number, nnnn, with the tool letter prefix, but without the letter suffix indicating the severity.
• warning, to treat all warnings as errors.

Related references

3.16 --diag_remark=tag[,tag,...] on page 3-52
3.17 --diag_style={arm|ide|gnu} on page 3-53
3.18 --diag_suppress=tag[,tag,...] on page 3-54
3.19 --diag_warning=tag[,tag,...] on page 3-55
### 3.16 --diag_remark=tag[,tag,...]

Sets diagnostic messages that have a specific tag to Remark severity.

**Syntax**

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

Where `tag` is a comma-separated list of diagnostic message numbers. This is the four-digit number, `nnnn`, with the tool letter prefix, but without the letter suffix indicating the severity.

**Related references**

3.15 --diag_error=tag[,tag,...] on page 3-51
3.17 --diag_style={arm|ide|gnu} on page 3-53
3.18 --diag_suppress=tag[,tag,...] on page 3-54
3.19 --diag_warning=tag[,tag,...] on page 3-55
3.17 --diag_style={arm|ide|gnu}

Specifies the display style for diagnostic messages.

Syntax

```
--diag_style=string
```

Where `string` is one of:

- **arm**: Display messages using the legacy Arm compiler 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 gcc.

Usage

- `--diag_style=gnu` matches the format reported by the GNU Compiler, gcc.
- `--diag_style=ide` matches the format reported by Microsoft Visual Studio.

Default

The default is `--diag_style=arm`.

Related references

3.15 --diag_error=tag[,tag,...] on page 3-51
3.16 --diag_remark=tag[,tag,...] on page 3-52
3.18 --diag_suppress=tag[,tag,...] on page 3-54
3.19 --diag_warning=tag[,tag,...] on page 3-55
3.18 \texttt{--diag_suppress=tag[,tag,...]}  

Suppresses diagnostic messages that have a specific tag.

Syntax

\texttt{--diag_suppress=tag[,tag,...]}

Where \texttt{tag} can be:

- A diagnostic message number to be suppressed. This is the four-digit number, \texttt{n Dunn}, with the tool letter prefix, but without the letter suffix indicating the severity.
- \texttt{error}, to suppress all errors that can be downgraded.
- \texttt{warning}, to suppress all warnings.

Related references

3.15 \texttt{--diag_error=tag[,tag,...]} on page 3-51
3.16 \texttt{--diag_remark=tag[,tag,...]} on page 3-52
3.17 \texttt{--diag_style={arm|ide|gnu}} on page 3-53
3.19 \texttt{--diag_warning=tag[,tag,...]} on page 3-55
3.19 `--diag_warning=tag[,tag,...]`

Sets diagnostic messages that have a specific tag to Warning severity.

**Syntax**

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

Where `tag` can be:

- A diagnostic message number to set to warning severity. This is the four-digit number, `nnnn`, with the tool letter prefix, but without the letter suffix indicating the severity.
- `error`, to set all errors that can be downgraded to warnings.

**Related references**

- 3.15 `--diag_error=tag[,tag,...]` on page 3-51
- 3.16 `--diag_remark=tag[,tag,...]` on page 3-52
- 3.17 `--diag_style={arm|ide|gnu}` on page 3-53
- 3.19 `--diag_warning=tag[,tag,...]` on page 3-55
3.20 **--disassemble**

Displays a disassembled version of the image to stdout. Disassembly is generated in armasm assembler syntax and not GNU assembler syntax.

**Usage**

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**.

**Example**

To disassemble the ELF file `infile.axf` for the Cortex-A7 processor and create a source file `outfile.asm`, enter:

```
fromelf --cpu=Cortex-A7 --disassemble --output=outfile.asm infile.axf
```

**Related references**

3.11 **--cpu=name** on page 3-45
3.23 **--emit=option[,option,...]** on page 3-59
3.40 **--interleave=option** on page 3-79
3.46 **--output=destination** on page 3-85
3.59 **--text** on page 3-99
3.21  --dump_build_attributes

Prints the contents of the build attributes section in raw hexadecimal form.

Restrictions

This option has no effect for AArch64 state inputs.

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
```

Related references

3.14  --decode_build_attributes on page 3-49
3.23  --emit=option[,option,...] on page 3-59
3.25  --extract_build_attributes on page 3-62
3.59  --text on page 3-99
### 3.22 -elf

Selects ELF output mode.

**Usage**

Use this option whenever you have to transform an ELF file into a slightly different ELF file. You also have to provide options to indicate how you want the file to be modified. The options are:

- `--debugonly`
- `--globalize`
- `--hide`
- `--hide_and_localize`
- `--in_place`
- `--hide`
- `--linkview` or `--no_linkview`. This option is deprecated.
- `--localize`
- `--rename`
- `--show`
- `--show_and_globalize`
- `--strip`
- `--show`
- `--symbolversions` or `--no_symbolversions`.

**Restrictions**

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

**Related references**

- 3.37 `--in_place` on page 3-75
- 3.46 `--output=destination` on page 3-85
- 3.57 `--strip=option[,option,...]` on page 3-96
3.23  --emit=option[,option,...]

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

Restrictions
You can use this option only in text mode.

Syntax
--emit=option[,option,...]
Where option is one of:

addresses
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
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
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
Prints contents of the data sections. It has the same effect as --text -d.

data_symbols
 Modifies the output information of data sections so that symbol definitions are interleaved.

dynamic_segment
Prints dynamic segment contents. It has the same effect as --text -y.

exception_tables
Decodes AArch32 exception table information for objects. It has the same effect as --text -e.

frame_directives
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
Prints the contents of the Global Offset Table (GOT) section.
heading_comments

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

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

relocation_tables

Prints relocation information. It has the same effect as --text -r.

string_tables

Prints the string tables. It has the same effect as --text -t.

summary

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

Prints symbols in disassembled code and data annotated with comments containing the respective property information.

Use this option with --disassemble.

symbol_tables

Prints the symbol and versioning tables. It has the same effect as --text -s.

whole_segments

Prints disassembled executables or shared libraries segment by segment even if it has a link view.

Use this option with --disassemble.

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

Related references

3.20 --disassemble on page 3-56
3.14 --decode_build_attributes on page 3-49
3.24 --expandarrays on page 3-61
3.59 --text on page 3-99
3.24 **--expandarrays**

Prints data addresses, including arrays that are expanded both inside and outside structures.

**Restrictions**

You can use this option with **--text -a** or with **--fieldoffsets**.

**Example**

The following example shows the output for a struct containing arrays when **--fieldoffsets** **--expandarrays** is specified:

```c
// foo.c
struct S {
    char A[8];
    char B[4];
};
struct S s;
struct S* get()
{
    return &s;
}

> armclang -target arm-arm-none-eabi -march=armv8-a -g -c foo.c
> fromelf --fieldoffsets --expandarrays foo.o

; Structure, S , Size 0xc bytes, from foo.c
S.A | EQU 0 ; array[8] of char
S.A[0] | EQU 0 ; char
S.A[1] | EQU 0x1 ; char
S.A[2] | EQU 0x2 ; char
S.A[3] | EQU 0x3 ; char
S.A[4] | EQU 0x4 ; char
S.A[5] | EQU 0x5 ; char
S.A[6] | EQU 0x6 ; char
S.A[7] | EQU 0x7 ; char
S.B | EQU 0x8 ; array[4] of char
S.B[0] | EQU 0x8 ; char
S.B[1] | EQU 0x9 ; char
S.B[2] | EQU 0xa ; char
S.B[3] | EQU 0xb ; char
; End of Structure S
END
```

**Related references**

3.26 **--fieldoffsets** on page 3-63
3.59 **--text** on page 3-99
---extract_build_attributes

Prints only the build attributes in a form that depends on the type of attribute.

**Usage**

Prints the build attributes in:
- Human-readable form for standard build attributes.
- Raw hexadecimal form for nonstandard build attributes.

**Restrictions**

This option has no effect for AArch64 state inputs.

**Example**

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

```bash
> armclang -c -mcpu=cortex-m7 --target=arm-arm-none-eabi -mfpu=vfpv3 hello.c -o hello.o
> fromelf --cpu=Cortex-M7 --extract_build_attributes hello.o
```

**Object/Image Build Attributes**

'aeabi' file build attributes:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
<td>43 32 2e 30 05 63 6f 72 74 78 2d 6d 67 00 06 0d 07 4d 08 00 02 0a 05 0e 00 11 01 12</td>
</tr>
<tr>
<td>0x000010</td>
<td>00 06 0d 07 4d 08 00 02 0a 05 0e 00 11 01 12</td>
</tr>
<tr>
<td>0x000020</td>
<td>04 14 01 15 01 17 03 18 01 19 01 1a 02 22 00 24</td>
</tr>
<tr>
<td>0x000030</td>
<td>01 26 01</td>
</tr>
</tbody>
</table>

**Related references**

- **3.14 --decode_build_attributes** on page 3-49
- **3.21 --dump_build_attributes** on page 3-57
- **3.23 --emit=option[,option,...]** on page 3-59
- **3.59 --text** on page 3-99
3.26 --fieldoffsets

Prints a list of armasm style assembly language EQU directives that equate C++ class or C structure field names to their offsets from the base of the class or structure.

Usage

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.

Note

The EQU directives cannot be used with the clang-integrated assembler. To use them, you must change them to GNU syntax.

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

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

Restrictions

This option:

• Requires that the object or image file has debug information.
• Can be used in text mode and with --expandarrays.

Examples

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:

```bash
fromelf --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:

```bash
fromelf --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:

```bash
fromelf --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:

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

The following is an example of the output, and includes name. and name...member that arise because of anonymous structs and unions:

```bash
; 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...Min| EQU 0        ; anonymous
| Box2...Min.x| EQU 0        ; short
| Box2...Min.y| EQU 0x2      ; short
```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box2...Max</td>
<td>EQU</td>
<td>0x4; Point2</td>
</tr>
<tr>
<td>Box2...Max.x</td>
<td>EQU</td>
<td>0x4; short</td>
</tr>
<tr>
<td>Box2...Max.y</td>
<td>EQU</td>
<td>0x6; short</td>
</tr>
<tr>
<td>; Warning: duplicate name (Box2..) present in (inputfile.cpp) and in (inputfile.cpp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>; please use the --qualify option</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Box2...</td>
<td>EQU</td>
<td>0; anonymous</td>
</tr>
<tr>
<td>Box2...Left</td>
<td>EQU</td>
<td>0; unsigned short</td>
</tr>
<tr>
<td>Box2...Top</td>
<td>EQU</td>
<td>0x2; unsigned short</td>
</tr>
<tr>
<td>Box2...Right</td>
<td>EQU</td>
<td>0x4; unsigned short</td>
</tr>
<tr>
<td>Box2...Bottom</td>
<td>EQU</td>
<td>0x6; unsigned short</td>
</tr>
<tr>
<td>; End of Structure Box2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>; Structure, MyClassHandle, Size 0x4 bytes, from inputfile.cpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyClassHandle.Handle</td>
<td>EQU</td>
<td>0; pointer to MyClass</td>
</tr>
<tr>
<td>; End of Structure MyClassHandle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>; Structure, Point2, Size 0x4 bytes, from defects.cpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point2.x</td>
<td>EQU</td>
<td>0; short</td>
</tr>
<tr>
<td>Point2.y</td>
<td>EQU</td>
<td>0x2; short</td>
</tr>
<tr>
<td>; End of Structure Point2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>; Structure, __fpos_t_struct, Size 0x10 bytes, from C:\Program Files\DS-5\bin.\include \stdio.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__fpos_t_struct.__pos</td>
<td>EQU</td>
<td>0; unsigned long long</td>
</tr>
<tr>
<td>__fpos_t_struct.__mbstate</td>
<td>EQU</td>
<td>0x8; anonymous</td>
</tr>
<tr>
<td>__fpos_t_struct.__mbstate.__state1</td>
<td>EQU</td>
<td>0x8; unsigned int</td>
</tr>
<tr>
<td>__fpos_t_struct.__mbstate.__state2</td>
<td>EQU</td>
<td>0xc; unsigned int</td>
</tr>
<tr>
<td>; End of Structure __fpos_t_struct</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related references

3.24 --expandarrays on page 3-61
3.48 --qualify on page 3-87
3.52 --select=select_options on page 3-91
3.59 --text on page 3-99

Related information

EQU
GET or INCLUDE

Miscellaneous directives
3.27 --fpu=list

Lists the FPU architectures that are supported by the --fpu=name option.

Deprecated options are not listed.

Related references
3.28 --fpu=name on page 3-66
### 3.28  --fpu=name

Specifies the target FPU architecture.

To obtain a full list of FPU architectures use the --fpu=list option.

**Syntax**

```
--fpu=name
```

Where *name* is the name of the target FPU architecture. Specify --fpu=list to list the supported FPU architecture names that you can use with --fpu=name.

The default floating-point architecture depends on the target architecture.

———  **Note**  ————

Software floating-point linkage is not supported for AArch64 state.

**Usage**

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

If you specify this option, it overrides any implicit FPU option that appears on the command line, for example, where you use the --cpu option.

Any FPU explicitly selected using the --fpu option always overrides any FPU implicitly selected using the --cpu option.

**Default**

The default target FPU architecture is derived from use of the --cpu option.

If the CPU you specify with --cpu has a VFP coprocessor, the default target FPU architecture is the VFP architecture for that CPU.

**Related references**

3.20 --disassemble on page 3-56
3.27 --fpu=list on page 3-65
3.38 --info=topic[,topic,...] on page 3-76
3.59 --text on page 3-99
3.29  --globalize=option[,option,...]

Converts the selected symbols to global symbols.

**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 values in one `option` followed by a comma-separated list of arguments.

**Restrictions**

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

**Related references**

3.22 `--elf` on page 3-58
3.31 `--hide=option[,option,...]` on page 3-69
### 3.30 --help

Displays a summary of the main command-line options.

**Default**

This is the default if you specify `fromelf` without any options or source files.

**Related references**

- `3.55 --show_cmdline` on page 3-94
- `3.60 --version_number` on page 3-101
- `3.63 --vsn` on page 3-104
### 3.31 --hide=option[,option,…]

Changes the symbol visibility property to mark selected symbols as hidden.

**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 values in one `option` followed by a comma-separated list of arguments.

**Restrictions**

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

**Related references**

3.22 `--elf` on page 3-58
3.53 `--show=option[,option,...]` on page 3-92
Changes the symbol visibility property to mark selected symbols as hidden, and converts the selected symbols to local symbols.

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 values in one `option` followed by a comma-separated list of arguments.

Restrictions

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

Related references

3.22 `--elf` on page 3-58
3.33  --i32

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.

Restrictions
The following restrictions apply:
• Not supported for AArch64 state.
• You cannot use this option with object files.
• You must use --output with this option.

Considerations when using --i32
If you convert an ELF image containing multiple load regions to a binary format, 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 only created when the load region describes code or data that is present in the ELF file. For example a load region containing only execution regions with ZI data in them does not result in an output file.

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

    fromelf --i32 --output=outfile.bin infile.axf

Related references
3.1 --base [object_file::load_region_ID=num on page 3-32
3.34 --i32combined on page 3-72
3.46 --output=destination on page 3-85
3.34 --i32combined

Produces Intel Hex-32 format output. It generates one output file for an image containing multiple load regions.

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

Restrictions
The following restrictions apply:
• Not supported for AArch64 state.
• You cannot use this option with object files.
• You must use --output with this option.

Considerations when using --i32combined

If you convert an ELF image containing multiple load regions to a binary format, fromelf creates an output directory named destination and generates one binary output file for all load regions in the input image. fromelf 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.

Example

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 --i32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

Related references

- 3.1 --base [object_file::]load_region_ID=num on page 3-32
- 3.33 --i32 on page 3-71
- 3.46 --output=destination on page 3-85
3.35 --ignore_section=option[,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.

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 `symbol_name` and `object_name` arguments
- Specify multiple values in one `option` followed by a comma-separated list of arguments.

Restrictions

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

Related references

- 3.8 --compare=option[,option,...] on page 3-41
- 3.36 --ignore_symbol=option[,option,...] on page 3-74
- 3.49 --relax_section=option[,option,...] on page 3-88
3.36 **--ignore_symbol=option[,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.

**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 all symbols with names matching `symbol_name`.
- `symbol_name`
  - All symbols with names matching `symbol_name`.

You can:

- Use wildcard characters `?` and `*` for symbolic names in `symbol_name` and `object_name` arguments
- Specify multiple values in one `option` followed by a comma-separated list of arguments.

**Restrictions**

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

**Related references**

- 3.8 `--compare=option[,option,...]` on page 3-41
- 3.35 `--ignore_section=option[,option,...]` on page 3-73
- 3.50 `--relax_symbol=option[,option,...]` on page 3-89
**3.37 --in_place**

Enables the translation of ELF members in an input file to overwrite the previous content.

**Restrictions**

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

**Example**

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

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

**Related references**

3.22 `--elf` on page 3-58
3.57 `--strip=option[,option,…]` on page 3-96
3.38  

--info=topic[,topic,…]  

Prints information about specific topics.

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 ________  
Not supported for AArch64 state.

________

**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.

________ Note ________  
Code related sizes also include the size of any execute-only code.

________

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.

________

Restrictions  
You can use this option only in text mode.

Related references  
3.59 --text on page 3-99
3.39 input_file

Specifies the ELF file or archive containing ELF files to be processed.

Usage

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.

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:

* Matches zero or more characters.

? Matched any single character.

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

archive.a(?str*)

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

Example

To convert all files in the archive beginning with s, and create a new archive, my_archive.a, containing the processed and unprocessed files, enter:

fromelf archive.a(s*.o) --output=my_archive.a

Related concepts
2.2 Examples of processing ELF files in an archive on page 2-23

Related references
3.2 --bin on page 3-34
3.6 --cad on page 3-38
3.7 --cadcombined on page 3-40
3.8 --compare=option[,option,...] on page 3-41
3.22 --elf on page 3-58
3.33 --i32 on page 3-71
3.34 --i32combined on page 3-72
3.37 --in_place on page 3-75
3.43 --m32 on page 3-82
3.44 --m32combined on page 3-83
3.46 --output=destination on page 3-85
3.59 --text on page 3-99
3.61 --vhx on page 3-102
3.40 --interleave=option

Inserts the original source code as comments into the disassembly if debug information is present.

**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**

Disables interleaving. 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.

**Usage**

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.

**Default**

The default is --interleave=none.

**Related references**

3.20 --disassemble on page 3-56
3.23 --emit=option[,option,...] on page 3-59
3.56 --source_directory==path on page 3-95
3.59 --text on page 3-99
3.41 --linkview, --no_linkview

Controls the section-level view from the ELF image.

Usage

--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.

----- Note -----
This option is deprecated.

Restrictions

The following restrictions apply:

• You must use --elf with --linkview and --no_linkview.

Example

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

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

Related references

3.22 --elf on page 3-58
3.47 --privacy on page 3-86
3.57 --strip=option[,option,...] on page 3-96

Related information

--privacy linker option
3.42  --localize=option[,option,...]

Converts the selected symbols to local symbols.

**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 values in one `option` followed by a comma-separated list of arguments.

**Restrictions**

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

**Related references**

3.22 `--elf` on page 3-58
3.31 `--hide=option[,option,...]` on page 3-69
3.43  --m32

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.

Restrictions
The following restrictions apply:

• Not supported for AArch64 state.
• You cannot use this option with object files.
• You must use --output with this option.

Considerations when using --m32
If you convert an ELF image containing multiple load regions to a binary format, 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 only created when the load region describes code or data that is present in the ELF file. For example a load region containing only execution regions with ZI data in them does not result in an output file.

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

```
fromelf --m32 --output=outfile.bin infile.axf
```

Related references
3.1 --base [(object_file::load_region_ID=)num on page 3-32
3.44 --m32combined on page 3-83
3.46 --output=destination on page 3-85
3.44 --m32combined

Produces Motorola 32-bit format (32-bit S-records) output. It generates one output file for an image containing multiple load regions.

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

Restrictions
The following restrictions apply:
• Not supported for AArch64 state.
• You cannot use this option with object files.
• You must use --output with this option.

Considerations when using --m32combined

If you convert an ELF image containing multiple load regions to a binary format, fromelf creates an output directory named destination and generates one binary output file for all load regions in the input image. fromelf 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.

Example

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 --m32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

Related references

3.1 --base [[object_file::]load_region_ID=num on page 3-32
3.43 --m32 on page 3-82
3.46 --output=destination on page 3-85
3.45  \texttt{--only=section\_name}

Filters the list of sections that are displayed in the main section-by-section output from \texttt{--text}. It does not affect any additional output after the main section-by-section output.

\textbf{Syntax}

\texttt{--only=section\_name}

Where \textit{section\_name} is the name of the section to be displayed.

You can:

\begin{itemize}
  \item Use wildcard characters \texttt{?} and \texttt{*} for a section name.
  \item Use multiple \texttt{--only} options to specify additional sections to display.
\end{itemize}

\textbf{Examples}

The following examples show how to use \texttt{--only}:

\begin{itemize}
  \item To display only the symbol table, \texttt{.symtab}, from the section-by-section output, enter:
    \begin{verbatim}
    fromelf --only=.symtab --text -s test.axf
    \end{verbatim}
  \item To display all \texttt{ER} sections, enter:
    \begin{verbatim}
    fromelf --only=ER? test.axf
    \end{verbatim}
  \item To display the \texttt{HEAP} section and all symbol and string table sections, enter:
    \begin{verbatim}
    fromelf --only=HEAP --only=.*tab --text -s -t test.axf
    \end{verbatim}
\end{itemize}

\textit{Related references}

3.59 \texttt{--text} on page 3-99
### 3.46 --output=destination

Specifies the name of the output file, or the name of the output directory if multiple output files are created.

**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.

**Usage**

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

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

```
fromelf --elf --strip=debug archive.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.

**Related references**

3.2 --bin on page 3-34
3.22 --elf on page 3-58
3.59 --text on page 3-99
3.47  --privacy

Modifies the output file to protect your code in images and objects that are delivered to third parties.

Usage

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.

Related references
3.57 --strip=option[,option,...] on page 3-96

Related information
--locals, --no_locals linker option
--privacy linker option
3.48 --qualify

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.

Usage

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.

Examples

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.

Related references

3.26 --fieldoffsets on page 3-63
3.49  --relax_section=option[,option,...]

Changes the severity of a compare report for the specified sections to warnings rather than errors.

Restrictions

You must use --compare with this option.

Syntax

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

Where option is one of:

object_name::
   All sections in ELF objects with a name matching object_name.

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

section_name
   All sections with a name matching section_name.

You can:

• Use wildcard characters ? and * for symbolic names in section_name and object_name arguments
• Specify multiple values in one option followed by a comma-separated list of arguments.

Related references

3.8 --compare=option[,option,...] on page 3-41
3.35 --ignore_section=option[,option,...] on page 3-73
3.50 --relax_symbol=option[,option,...] on page 3-89
3.50  --relax_symbol=option[,option,…]

Changes the severity of a compare report for the specified symbols to warnings rather than errors.

Restrictions
You must use --compare with this option.

Syntax

--relax_symbol=option[,option,…]

Where option is one of:

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

object_name::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 name matching symbol_name.

You can:
• Use wildcard characters ? and * for symbolic names in symbol_name and object_name arguments
• Specify multiple values in one option followed by a comma-separated list of arguments.

Related references
3.8 --compare=option[,option,...] on page 3-41
3.36 --ignore_symbol=option[,option,...] on page 3-74
3.49 --relax_section=option[,option,...] on page 3-88
3.51  \texttt{--rename=option[,option,...]}  

Renames the specified symbol in an output ELF object.

\textbf{Restrictions}

You must use \texttt{--elf} and \texttt{--output} with this option.

\textbf{Syntax}

\texttt{--rename=option[,option,...]}

Where \texttt{option} is one of:

\texttt{object_name::old_symbol_name=new_symbol_name}

This replaces all symbols in the ELF object \texttt{object_name} that have a symbol name matching \texttt{old_symbol_name}.

\texttt{old_symbol_name=new_symbol_name}

This replaces all symbols that have a symbol name matching \texttt{old_symbol_name}.

You can:

\begin{itemize}
\item Use wildcard characters \texttt{?} and \texttt{*} for symbolic names in \texttt{old_symbol_name}, \texttt{new_symbol_name}, and \texttt{object_name} arguments.
\item Specify multiple values in one \texttt{option} followed by a comma-separated list of arguments.
\end{itemize}

\textbf{Example}

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

\begin{verbatim}
fromelf --elf --rename=clock=myclock --output=mytimer.axf timer.axf
\end{verbatim}

\textbf{Related references}

3.22 \texttt{--elf} on page 3-58
3.46 \texttt{--output=destination} on page 3-85
3.52 --select=select_options

When used with --fieldoffsets or --text -a options, displays only those fields that match a specified pattern list.

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.

Usage

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

Example

The output from the --fieldoffsets option might include the following data structure:

| structure.f1 | EQU 0 ; int16_t
| structure.f2 | EQU 0x2 ; int16_t
| structure.f3 | EQU 0x4 ; int16_t
| structure.f11| EQU 0x6 ; int16_t
| structure.f21| EQU 0x8 ; int16_t
| structure.f31| EQU 0xA ; int16_t
| structure.f111| EQU 0xC ; int16_t

To output only those fields that start with f1, enter:

```
fromelf --select=structure.f1* --fieldoffsets infile.axf
```

This produces the output:

```
| structure.f1 | EQU 0 ; int16_t
| structure.f11| EQU 0x6 ; int16_t
| structure.f111| EQU 0xC ; int16_t

END
```

Related references

3.26 --fieldoffsets on page 3-63
3.59 --text on page 3-99
3.53  --show=option[,option,…]

Changes the symbol visibility property of the selected symbols, to mark them with default visibility.

**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 values in one `option` followed by a comma-separated list of arguments.

**Restrictions**

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

**Related references**

3.22 `--elf` on page 3-58
3.31 `--hide=option[,option,...]` on page 3-69
3.54  --show_and_globalize=option[,option,...]

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

**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 values in one *option* followed by a comma-separated list of arguments.

**Restrictions**

You must use --elf with this option.

**Related references**

3.22 --elf on page 3-58
3.55  --show_cmdline

Outputs the command line used by the ELF file converter.

Usage
Shows the command line after processing by the ELF file converter, and can be useful to check:
• The command line a build system is using.
• How the ELF file converter is interpreting the supplied command line, for example, the ordering of
  command-line options.

The commands are shown normalized, and the contents of any via files are expanded.

The output is sent to the standard error stream (stderr).

Related references
3.62 --via=file on page 3-103
3.56  --source_directory=path

Explicitly specifies the directory of the source code.

Syntax

--source_directory=path

Usage

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.

Related references

3.40 --interleave=option on page 3-79
3.57 --strip=option[,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.

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`.

- `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.

---------

Restrictions

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

Example

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

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

Related references

3.22 --elf on page 3-58
3.41 --linkview, --no_linkview on page 3-80
3.47 --privacy on page 3-86

Related information

About mapping symbols
--locals, --no_locals linker option
--privacy linker option
3.58 --symbolversions, --no_symbolversions

Turns off the decoding of symbol version tables.

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

Related information
About symbol versioning
Base Platform ABI for the ARM Architecture
3.59 --text

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

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 fields in a data structure.
  
  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**

  Disassembly is generated in `armasm` assembler syntax and not GNU assembler syntax.
  
  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.

  **Note**

  Not supported for AArch64 state.

- `-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.
Usage
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.

Use the --only option to filter the list of sections.

Examples
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 --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 -a --select=* infile.axf
```

• To produce a text file containing all of the structure addresses in infile.axf but none of the global or static data variable information, enter:

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

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

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

• To produce a text file containing all of the global or static data variable information in infile.axf but none of the structure addresses, enter:

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

• To output only the .symtab section information in infile.axf, enter:

```
fromelf --only .symtab -s --output=symtab.txt infile.axf
```

Related tasks
2.6 Using fromelf to find where a symbol is placed in an executable ELF image on page 2-28

Related references
3.11 --cpu=name on page 3-45
3.20 --disassemble on page 3-56
3.23 --emit=option[,option,...] on page 3-59
3.24 --expandarrays on page 3-61
3.38 --info=topic[,topic,...] on page 3-76
3.40 --interleave=option on page 3-79
3.45 --only=section_name on page 3-84
3.46 --output=destination on page 3-85
3.52 --select=select_options on page 3-91
3.64 -w on page 3-105

Related information
Linker options for getting information about images
3.60  --version_number

Displays the version of fromelf you are using.

Usage
The ELF file converter displays the version number in the format Mmmuuxx, where:

• M is the major version number, 6.
• mm is the minor version number.
• uu is the update number.
• xx is reserved for Arm internal use. You can ignore this for the purposes of checking whether the current release is a specific version or within a range of versions.

Related references
3.30 --help on page 3-68
3.63 --vsn on page 3-104
3.61  --vhx

Produces Byte oriented (Verilog Memory Model) hexadecimal format output.

Usage

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.

Restrictions

The following restrictions apply:

- You cannot use this option with object files.
- You must use --output with this option.

Considerations when using --vhx

If you convert an ELF image containing multiple load regions to a binary format, 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 only created when the load region describes code or data that is present in the ELF file. For example a load region containing only execution regions with ZI data in them does not result in an output file.

Examples

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

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

Related references

3.46 --output=destination on page 3-85
3.66 --widthxbanks on page 3-107
3.62  --via=file

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

**Syntax**

--via=*filename*

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

**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.

**Related concepts**

4.1 Overview of via files on page 4-110

**Related references**

4.2 Via file syntax rules on page 4-111
3.63  --vsn

Displays the version information and the license details.

Note

`--vsn` is intended to report the version information for manual inspection. The `Component` line indicates the release of Arm Compiler you are using. If you need to access the version in other tools or scripts, for example in build scripts, use the output from `--version_number`.

Example

```
> fromelf --vsn
Product: ARM Compiler N.n
Component: ARM Compiler N.n
Tool: fromelf [tool_id]
license_type
Software supplied by: ARM Limited
```

Related references

3.30  `--help` on page 3-68
3.60  `--version_number` on page 3-101
3.64 -w

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

Usage

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

Example

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

```
========================================================================
** ELF Header Information
  
  ** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXCINSTR] 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: 0x00000000   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
```
3.65 --wide64bit

Causes all addresses to be displayed with a width of 64 bits.

**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.

*Related references*

3.39 input_file on page 3-77
3.66 \texttt{--widthxbanks}

Outputs multiple files for multiple memory banks.

**Syntax**

\texttt{--widthxbanks}

Where:

- \texttt{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.

- \texttt{width} is the width of memory in the target memory system (8-bit, 16-bit, 32-bit, or 64-bit).

Valid configurations are:

- \texttt{--8x1}
- \texttt{--8x2}
- \texttt{--8x4}
- \texttt{--16x1}
- \texttt{--16x2}
- \texttt{--32x1}
- \texttt{--32x2}
- \texttt{--64x1}

**Usage**

\texttt{fromelf} uses the last specified configuration if more than one configuration is specified.

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

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

  \begin{verbatim}
  fromelf --cpu=8-A.32 --vhx --8x2 test.axf --output=test.txt
  \end{verbatim}

  This generates two files named \texttt{test0.txt} and \texttt{test1.txt}.

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

\begin{verbatim}
fromelf --cpu=8-A.32 --vhx --8x2 multiload.axf --output=regions/
\end{verbatim}

This might produce the following files in the \texttt{regions} directory:

<table>
<thead>
<tr>
<th>EXEC_ROM0</th>
<th>EXEC_ROM1</th>
<th>RAM0</th>
<th>RAM1</th>
</tr>
</thead>
</table>

The memory width specified by \texttt{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:

- \texttt{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:

<table>
<thead>
<tr>
<th>00</th>
<th>00</th>
<th>2D</th>
<th>00</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>00</th>
<th>2C</th>
</tr>
</thead>
</table>
fromelf --vhx --16x2 test.axf --output=file produces two files (file0 and file1). Each file contains lines of two bytes, for example:

```
0000
002D
002C
```

**Restrictions**

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

**Related references**

- 3.2 **--bin** on page 3-34
- 3.46 **--output=destination** on page 3-85
- 3.61 **--vhx** on page 3-102
Chapter 4
Via File Syntax

Describes the syntax of via files accepted by the armasm, armlink, fromelf, and armar tools.

It contains the following sections:

• 4.1 Overview of via files on page 4-110.
• 4.2 Via file syntax rules on page 4-111.
4.1 Overview of via files

Via files are plain text files that allow you to specify command-line arguments and options for the `armasm`, `armlink`, `fromelf`, and `armar` tools.

Typically, you 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`. Therefore, you can call multiple nested via files from within a via file.

Via file evaluation

When you invoke the `armasm`, `armlink`, `fromelf`, or `armar`, the tool:

1. Replaces the first specified `--via via_file` argument with the sequence of argument words that are 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 that you specify them. Each via file is processed completely, including any nested via files contained in that file, before processing the next via file.

Related references

4.2 Via file syntax rules on page 4-111
3.62 --via=file on page 3-103
4.2 Via file syntax rules

Via files must conform to some 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:
  -\texttt{--vhx \ --8x2} (two words)
  -\texttt{--vhx--8x2} (one word)
- The end of a line is treated as whitespace, for example:
  -\begin{verbatim}
  \texttt{--vhx \--8x2}
  \end{verbatim}

This is equivalent to:
  -\texttt{--vhx \ --8x2}

- 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:
  -\texttt{--output C:\My Project\output.txt} (three words)
  -\texttt{--output "C:\My Project\output.txt"} (two words)

Use apostrophes to delimit words that contain quotes, for example:
  -\texttt{-DNAME='"ARM Compiler"'} (one word)

- Characters enclosed in parentheses are treated as a single word, for example:
  -\texttt{--option(x, y, z)} (one word)
  -\texttt{--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:
  -\texttt{--output"C:\Project\output.txt"}

This is treated as the single word:
  -\texttt{--outputC:\Project\output.txt}

- Lines beginning with a semicolon (;) or a hash (#) character as the first nonwhitespace character are comment lines. A semicolon or hash character that appears anywhere else in a line is not treated as the start of a comment, for example:
  -\texttt{-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.

\textit{Related concepts}

4.1 Overview of via files on page 4-110

\textit{Related references}

3.62 --via=file on page 3-103