ARM® Compiler

Version 6.8

armar User Guide

ARM

ARM 100072_0608_00_en

ARM® Compiler

armar User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	14 March 2014	Non-Confidential	ARM Compiler v6.00 Release
B	15 December 2014	Non-Confidential	ARM Compiler v6.01 Release
C	30 June 2015	Non-Confidential	ARM Compiler v6.02 Release
D	18 November 2015	Non-Confidential	ARM Compiler v6.3 Release
E	24 February 2016	Non-Confidential	ARM Compiler v6.4 Release
F	29 June 2016	Non-Confidential	ARM Compiler v6.5 Release
G	04 November 2016	Non-Confidential	ARM Compiler v6.6 Release
0607-00	05 April 2017	Non-Confidential	ARM Compiler v6.7 Release. Document numbering scheme has changed.
0608-00	30 July 2017	Non-Confidential	ARM Compiler v6.8 Release.

Non-Confidential Proprietary Notice

ARM Limited. Company 02557590 registered in England.

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

http://www.arm.com

Contents
ARM® Compiler armar User Guide

Preface

This preface introduces the *ARM® Compiler armar User Guide*.

It contains the following:

- *About this book* on page 7.

About this book

ARM® Compiler armar User Guide provides information on how to use the `armar` utility.

Using this book

This book is organized into the following chapters:

Chapter 1 Overview of the ARM Librarian

Gives an overview of the ARM librarian, `armar`, provided with ARM Compiler.

Chapter 2 armar Command-line Options

Describes the command-line options of the ARM librarian, `armar`.

Chapter 3 Via File Syntax

Describes the syntax of via files accepted by `armar`.

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 armar User Guide*.
- The number ARM 100072_0608_00_en.
- 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*.
- *Support and Maintenance*.
- *ARM Glossary*.

Chapter 1
Overview of the ARM Librarian

Gives an overview of the ARM librarian, `armar`, provided with ARM Compiler.

It contains the following sections:

1.1 About the ARM librarian

The ARM librarian, `armar`, enables you to collect and maintain sets of ELF object files in standard format `ar` libraries.

You can pass these libraries to the linker in place of several ELF object files.

With `armar` you can:

- Create new libraries.
- Add files to a library.
- Replace individual files in a library.
- Replace all files in a library with specified files in a single operation.
- Control the placement of files in a library.
- Display information about a specified library. For example, list all members in a library.

A timestamp is also associated with each file that is added or replaced in a library.

Note

When you create, add, or replace object files in a library, `armar` creates a symbol table by default. However, debug symbols are not included by default.

Related references

2.8 --debug_symbols on page 2-23.

Related information

--library=name linker option.

--libpath=pathlist linker option.

--library_type=lib linker option.

--userlibpath=pathlist linker option.

1.2 Considerations when working with library files

There are some considerations you must be aware of when working with library files.

Be aware of the following:

- A library differs from a shared object or *dynamically linked library* (DLL) in that:
  - Symbols are imported from a shared object or DLL.
  - Code or data for symbols is extracted from an archive into the file being linked.
- Linking with an object library file might not produce the same results as linking with all the object files collected into the object library file. This is because the linker processes the input list and libraries differently:
  - Each object file in the input list appears in the output unconditionally, although unused areas are eliminated if the `armlink --remove` option is specified.
  - A member of a library file is only included in the output if it is referred to by an object file or a previously processed library file.

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

Related information

--remove, --no_remove linker option.

1.3 armar command-line syntax

The `armar` command has options to specify how to process files and libraries.

Syntax

`armar` *`options archive`* [*`file_list`*]

`options`

`armar` command-line options.

`archive`

The filename of the library. A library file must always be specified.

`file_list`

The list of files to be processed.

Related references

2.1 archive on page 2-16.
2.15 file_list on page 2-30.

1.4 Option to get help on the armar 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 source files.

Example

To display the help information, enter:

```
armar --help
```

Chapter 2
armar Command-line Options

Describes the command-line options of the ARM librarian, `armar`.

It contains the following sections:

2.1 archive

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

Note

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

Related references

2.15 file_list on page 2-30.

2.2 -a pos_name

Places new files in the library after the specified library member.

Syntax

`-a=`*`pos_name`*

Where *`pos_name`* is the name of a file in the library.

Usage

The effect of this option is negated if you include `-b` (or `-i`) on the same command line.

Example

To add or replace files `obj3.o` and `obj4.o` immediately after `obj2.o` in `mylib.a`, enter:

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

Related references

2.3 -b pos_name on page 2-18.
2.17 -i pos_name on page 2-32.
2.18 -m pos_name on page 2-33.
2.22 -r on page 2-37.

2.3 -b pos_name

Places new files in the library before the specified library member.

Syntax

`-b=`*`pos_name`*

Where *`pos_name`* is the name of a file in the library.

Usage

This option takes precedence if you include `-a` on the same command line.

Related references

2.2 -a pos_name on page 2-17.
2.17 -i pos_name on page 2-32.
2.18 -m pos_name on page 2-33.
2.22 -r on page 2-37.

2.4 -c

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

2.5 -C

Instructs the librarian not to replace existing files with like-named files when performing extractions.

Usage

Use this option with `-T` to prevent truncated filenames from replacing files with the same prefix.

An error message is displayed if the file to be extracted already exists in the current location.

Related references

2.27 -T on page 2-42.
2.33 -x on page 2-48.

2.6 --create

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

Usage

With the `--create` option specify the list of object files, either:

- Directly on the command-line.
- In a via file.

Note

If the library already exists, the previous contents are deleted.

You can use this option together with the following compatible command-line options:

- `-c`
- `--diag_style`
- `-n`
- `-v`
- `--via`.

Note

Other options can also create a new library in some circumstances. For example, using the `-r` option with a library that does not exist.

Examples

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

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

To create a new library containing the files listed in a via file, enter:

```
armar --create mylib.a --via myobject.via
```

Related references

2.15 file_list on page 2-30.

2.7 -d

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

You can use this option in conjunction with other compatible command-line options.

Example

To delete the files `file1.o` and `file2.o` from the `mylib.a` library, enter:

```
armar -d mylib.a file1.o,file2.o
```

Related references

2.15 file_list on page 2-30.

2.8 --debug_symbols

By default, debug symbols are excluded from an archive. Use `--debug_symbols` to include debug symbols in the archive.

Related concepts

1.1 About the ARM librarian on page 1-10.

2.9 --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

2.10 --diag_remark=tag[,tag,...] on page 2-25.
2.11 --diag_style={arm|ide|gnu} on page 2-26.
2.12 --diag_suppress=tag[,tag,...] on page 2-27.
2.13 --diag_warning=tag[,tag,...] on page 2-28.

2.10 --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

2.9 --diag_error=tag[,tag,...] on page 2-24.
2.11 --diag_style={arm|ide|gnu} on page 2-26.
2.12 --diag_suppress=tag[,tag,...] on page 2-27.
2.13 --diag_warning=tag[,tag,...] on page 2-28.

2.11 --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 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

2.9 --diag_error=tag[,tag,...] on page 2-24.
2.10 --diag_remark=tag[,tag,...] on page 2-25.
2.12 --diag_suppress=tag[,tag,...] on page 2-27.
2.13 --diag_warning=tag[,tag,...] on page 2-28.

2.12 --diag_suppress=tag[,tag,…]

Suppresses diagnostic messages that have a specific tag.

Syntax

`--diag_suppress=`*`tag[,tag,…]`*
Where *`tag`* can be:

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

Related references

2.9 --diag_error=tag[,tag,...] on page 2-24.
2.10 --diag_remark=tag[,tag,...] on page 2-25.
2.11 --diag_style={arm|ide|gnu} on page 2-26.
2.13 --diag_warning=tag[,tag,...] on page 2-28.

2.13 --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

2.9 --diag_error=tag[,tag,...] on page 2-24.
2.10 --diag_remark=tag[,tag,...] on page 2-25.
2.11 --diag_style={arm|ide|gnu} on page 2-26.
2.12 --diag_suppress=tag[,tag,...] on page 2-27.

2.14 --entries

Lists all object files in the library that have an entry point. You can use the `armasm` `ENTRY` directive to specify an entry point in legacy ARM syntax assembler code.

Usage

The format for the listing is:

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

Example

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

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

Related references

2.25 --sizes on page 2-40.
2.35 --zt on page 2-50.

Related information

ENTRY.
Miscellaneous directives.

2.15 file_list

A space-separated list of ELF-compliant files, such as ELF objects and ELF libraries.

Usage

Each file must be fully specified by its path and name. The path can be absolute, relative to drive and root, or relative to the current directory.

Note

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

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

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

2.16 --help

Displays a summary of the main command-line options.

Default

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

Related references

2.30 --version_number on page 2-45.
2.32 --vsn on page 2-47.

2.17 -i pos_name

Places new files in the library before the specified library member.

Syntax

`-i=pos_name`

Where *pos_name* is the name of a file in the library.

This is equivalent to `-b pos_name`

Related references

2.2 -a pos_name on page 2-17.
2.3 -b pos_name on page 2-18.
2.18 -m pos_name on page 2-33.
2.22 -r on page 2-37.

2.18 -m pos_name

Moves files in a library to a specified position.

Syntax

`-m=`*`pos_name`*

Where *`pos_name`* is the name of a file in the library.

Usage

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

Example

To move the file `file1.o` to a new location after `file2.o` in the `mylib.a` library, enter:

```
armar -m -a file2.o mylib.a file1.o
```

Related references

2.2 -a pos_name on page 2-17.
2.3 -b pos_name on page 2-18.
2.17 -i pos_name on page 2-32.

2.19 -n

Suppresses the creation of a symbol table in the library.

Usage

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

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

Example

To create a library without a symbol table, enter:

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

Related references

2.23 -s on page 2-38.

2.20 --new_files_only

Updates an object file in the archive only if the new object has a later timestamp.

Usage

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

Related references

2.22 -r on page 2-37.
2.28 -u on page 2-43.

2.21 -p

Prints the contents of source files in a library to `stdout`.

Note

The files must be text files.

Example

To display the contents of `file1.c` in `mylib.a`, enter:

```
armar -p mylib.a file1.c
```

Related references

2.26 -t on page 2-41.

2.22 -r

Replaces, or adds, files in the specified library.

Usage

If the library does not exist, a new library file is created and a diagnostic message is written to standard error. You can use this option in conjunction with other compatible command-line options.

`-q` is an alias for `-r`.

If no files are specified and the library exists, the results are undefined. Files that replace existing files do not change the order of the library.

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

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

Examples

To add or replace `obj1.o`, `obj2.o`, and `obj3.o` files in a library, enter:

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

To replace files with names beginning with k in a library, and only if the file in the library is older than the specified file, enter:

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

Related references

2.2 -a pos_name on page 2-17.
2.3 -b pos_name on page 2-18.
2.17 -i pos_name on page 2-32.
2.28 -u on page 2-43.
2.15 file_list on page 2-30.

2.23 -s

Creates a symbol table in the library.

Usage

This option is useful for libraries that have been created:

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

Note

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

Example

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

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

Related references

2.19 -n on page 2-34.
2.34 --zs on page 2-49.

2.24 --show_cmdline

Outputs the command line used by the librarian.

Usage

Shows the command line after processing by the librarian, and can be useful to check:

- The command line a build system is using.
- How the librarian 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`).

Example

To show how `armar` processes the command-line options for the replacement of file `obj1.o` in `mylib.a`, enter:

```
> armar --show_cmdline -r mylib.a obj1.o
[armar --show_cmdline -r mylib.a obj1.o]
```

Related references

2.31 --via=filename on page 2-46.

2.25 --sizes

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

Example

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

```
> armar --sizes mylib.a
Code    RO Data    RW data    ZI Data    Debug   Object Name
 464          0          0          0     8612   app_1.o
3356          0          0      10244    11848   app_2.o
3820          0          0      10244    20460   TOTAL
```

Related references

2.14 --entries on page 2-29.
2.35 --zt on page 2-50.

2.26 -t

Prints a table of contents for the library.

Usage

The files specified by *file_list* are included in the written list. If *file_list* is not specified, all files in the library are included in the order of the archive.

Examples

To display the table of contents of `mylib.a`, enter:

```
> armar -t mylib.a
app_1.o
app_2.o
```

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

```
> armar -tv mylib.a
rw-rw-rw-     0/     0   7512 Jun 22 11:19 2009 app_1.o (offset    736)
rw-rw-rw-     0/     0   1452 May 19 16:25 2009 app_2.o (offset   8308)
```

Related references

2.29 -v on page 2-44.
2.15 file_list on page 2-30.

2.27 -T

Enables truncation of filenames when extracted files have library names that are longer than the file system can support.

Usage

By default, extracting a file with a name that is too long is an error. A diagnostic message is written and the file is not extracted.

Be aware that if multiple files in the library have the same truncated name, each subsequent file that is extracted overwrites the previously extracted file with that name. To prevent this, use the `-C` option.

Related references

2.5 -C on page 2-20.
2.33 -x on page 2-48.

2.28 -u

Updates older files in the specified archive.

Usage

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

Related references

2.20 --new_files_only on page 2-35.
2.22 -r on page 2-37.

2.29 -v

Gives verbose output.

Usage

The output depends on what other options are used:

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

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

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

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

Related references

2.7 -d on page 2-22.
2.21 -p on page 2-36.
2.22 -r on page 2-37.
2.26 -t on page 2-41.
2.33 -x on page 2-48.

2.30 --version_number

Displays the version of `armar` you are using.

Usage

The librarian 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

2.16 --help on page 2-31.

2.32 --vsn on page 2-47.

2.31 --via=filename

Reads an additional list of input filenames and librarian 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 librarian command line. The `--via` options can also be included within a via file.

Related concepts

3.1 Overview of via files on page 3-52.

2.32 --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

Example output:

```
> armar --vsn
Product: ARM Compiler N.n
Component: ARM Compiler N.n
Tool: armar [tool_id]
```

Related references

2.16 --help on page 2-31.

2.30 --version_number on page 2-45.

2.33 -x

Extracts the files specified in *file_list* from the library to the current directory.

Usage

The contents of the library are not changed. If no file operands are given, all files in the library are extracted.

Be aware that if the name of a file in the library is longer than the file system can support, an error is displayed and the file is not extracted. To extract files with long filenames, use the `-T` option to truncate the names of files that are too long.

The files are extracted to the current location.

You can use this option in conjunction with other compatible command-line options.

Example

To extract the files `file1.o` and `file2.o` from the `mylib.a` library in the directory `C:\temp` to `C:\temp\obj`, enter:

```
cd \temp\obj
```

```
armar -x ..\mylib.a file1.o,file2.o
```

Related references

2.27 -T on page 2-42.
2.15 file_list on page 2-30.

2.34 --zs

Displays the symbol table for all files in the library.

Example

To list the symbol table in `mylib.a`, enter:

```
> armar --zs mylib.a
__ARM_use_no_argv    from hello.o    at offset    412
main                 from hello.o    at offset    412
__ARM_use_no_argv    from test.o     at offset   7960
main                 from test.o     at offset   7960
__ARM_use_no_argv    from hello_ltcg.o at offset  11408
main                 from hello_ltcg.o at offset  11408
__ARM_use_no_argv    from h1.o       at offset  18532
main                 from h1.o       at offset  18532
__ARM_use_no_argv    from fncalls.o  at offset   2072
add                  from fncalls.o  at offset   2072
main                 from fncalls.o  at offset   2072
get_stacksize        from get_stacksize.o at offset   9672
altstack             from get_stacksize.o at offset   9672
__ARM_use_no_argv    from s.o        at offset  13068
main                 from s.o        at offset  13068
altstack             from s.o        at offset  13068
_Z1fv                from t.o        at offset  17064
_ZN1T1fEi            from t.o        at offset  17064
```

Related references

2.19 -n on page 2-34.
2.23 -s on page 2-38.

2.35 --zt

Lists both the member sizes and entry points for all files in the library.

Example

To list the member sizes and entry points for all files in `mylib.a`, enter:

```
>armar --zt mylib.a

     Code    RO Data    RW Data    ZI Data      Debug   Object Name
      838          0          0          0          0   hello.o
       16          0          0          0       2869   fncalls.o
      893          0          0          0          0   test.o
      962          0          0          0          0   get_stacksize.o
      838          0          0          0          0   hello_ltcg.o
        8          0          0         80          0   s.o
       56          0         50          0          0   strcopy.o
        4          0         44          0        168   emit-relocs-1a.o
       36          8          0          0         84   t.o
      838          0          0          0          0   h1.o
     4489          8         94         80       3121   TOTAL

ENTRY at offset 0 in section StrCopy of strcopy.o
ENTRY at offset 0 in section StrCopy of emit-relocs-1a.o
```

Related references

2.14 --entries on page 2-29.
2.25 --sizes on page 2-40.

Chapter 3
Via File Syntax

Describes the syntax of via files accepted by `armar`.

It contains the following sections:

3.1 Overview of via files

Via files are plain text files that allow you to specify librarian command-line arguments and options.

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`. This means that you can call multiple nested via files from within a via file.

Via file evaluation

When the librarian is invoked it:

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

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

Related references

3.2 Via file syntax rules on page 3-53.
2.31 --via=filename on page 2-46.

3.2 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:

  `-d -v` (two words)

  `-d-v` (one word)
- The end of a line is treated as whitespace, for example:

  ```
  -d
  -v
  ```

  This is equivalent to:

  `-d -v`
- 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:

  `--via C:\My Project\viafile` (three words)

  `--via "C:\My Project\viafile"` (two words)

  Use apostrophes to delimit words that contain quotes, for example:

  `-DNAME='"ARM Compiler"'` (one word)
- Characters enclosed in parentheses are treated as a single word, for example:

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

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

  `--via"C:\Project\viafile"`

  This is treated as the single word:

  `--viaC:\Project\viafile`
- 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:

  ```
  -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.

Related concepts

3.1 Overview of via files on page 3-52.

Related references

2.31 --via=filename on page 2-46.