ARM® Compiler toolchain v4.1 for µVision

Linker Reference

ARM®

ARM Compiler toolchain v4.1 for µVision
Linker Reference

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
December 2008	A	Non-Confidential	Release for RVCT v4.0 for µVision
June 2011	B	Non-Confidential	Release for ARM Compiler toolchain v4.1 for µVision

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents

ARM Compiler toolchain v4.1 for µVision Linker Reference

Chapter 1
Conventions and feedback

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

Typographical conventions

The following typographical conventions are used:

`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`mono`space (mono underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

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

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on content

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

- the title
- the number, ARM DUI 0458B
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

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

Other information

- ARM Product Manuals, `http://www.keil.com/support/man_arm.htm`
- Keil Support Knowledgebase, `http://www.keil.com/support/knowledgebase.asp`
- Keil Product Support, `http://www.keil.com/support/`
- ARM Glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
Linker command-line options

The following topics describe the command-line options supported by the linker, `armlink`:

- *--datacompressor=opt* on page 2-29
- *--debug, --no_debug* on page 2-30
- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--eager_load_debug, --no_eager_load_debug* on page 2-36
- *--edit=file_list* on page 2-37
- *--emit_debug_overlay_relocs* on page 2-38
- *--emit_debug_overlay_section* on page 2-39
- *--emit_non_debug_relocs* on page 2-40
- *--emit_relocs* on page 2-41
- *--entry=location* on page 2-42
- *--errors=file* on page 2-44
- *--exceptions, --no_exceptions* on page 2-45
- *--exceptions_tables=action* on page 2-46
- *--export_dynamic, --no_export_dynamic* on page 2-47
- *--feedback=file* on page 2-48
- *--feedback_image=option* on page 2-49
- *--feedback_type=type* on page 2-50
- *--filtercomment, --no_filtercomment* on page 2-51
- *--fini=symbol* on page 2-52
- *--first=section_id* on page 2-53
- *--force_explicit_attr* on page 2-54
- *--force_so_throw, --no_force_so_throw* on page 2-55
- *--fpu=list* on page 2-56
- *--fpu=name* on page 2-57
- *--help* on page 2-58
- *--info=topic[,topic,...]* on page 2-59
- *--info_lib_prefix=opt* on page 2-62
- *--init=symbol* on page 2-63
- *--inline, --no_inline* on page 2-64
- *--inlineveneer, --no_inlineveneer* on page 2-65
- *input-file-list* on page 2-66
- *--keep=section_id* on page 2-68
- *--keep_protected_symbols* on page 2-70
- *--largeregions, --no_largeregions* on page 2-71
- *--last=section_id* on page 2-72
- *--ldpartial* on page 2-73
- *--legacyalign, --no_legacyalign* on page 2-74
- *--libpath=pathlist* on page 2-75
- *--library_type=lib* on page 2-76
- *--list=file* on page 2-77
- *--list_mapping_symbols, --no_list_mapping_symbols* on page 2-78
- *--load_addr_map_info, --no_load_addr_map_info* on page 2-79
- *--locals, --no_locals* on page 2-80

- *--ltcg* on page 2-81
- *--mangled, --unmangled* on page 2-82
- *--map, --no_map* on page 2-83
- *--match=crossmangled* on page 2-84
- *--max_veneer_passess=value* on page 2-85
- *--max_visibility=type* on page 2-86
- *--merge, --no_merge* on page 2-87
- *--muldefweak, --no_muldefweak* on page 2-88
- *--output=file* on page 2-89
- *--override_visibility* on page 2-90
- *--pad=num* on page 2-91
- *--paged* on page 2-92
- *--pagesize=pagesize* on page 2-93
- *--partial* on page 2-94
- *--piveneer, --no_piveneer* on page 2-95
- *--predefine="string"* on page 2-96
- *--privacy* on page 2-98
- *--reduce_paths, --no_reduce_paths* on page 2-99
- *--ref_cpp_init, --no_ref_cpp_init* on page 2-100
- *--reloc* on page 2-101
- *--remarks* on page 2-102
- *--remove, --no_remove* on page 2-103
- *--ro_base=address* on page 2-104
- *--ropi* on page 2-105
- *--rosplit* on page 2-106
- *--rw_base=address* on page 2-107
- *--rwpi* on page 2-108
- *--scanlib, --no_scanlib* on page 2-109
- *--scatter=file* on page 2-110
- *--section_index_display=type* on page 2-111
- *--show_cmdline* on page 2-112
- *--show_full_path* on page 2-113
- *--show_parent_lib* on page 2-114
- *--show_sec_idx* on page 2-115
- *--sort=algorithm* on page 2-116
- *--split* on page 2-118
- *--startup=symbol, --no_startup* on page 2-119
- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127
- *--symbolic* on page 2-128
- *--symbols, --no_symbols* on page 2-129

- *--symdefs=file* on page 2-130
- *--tailreorder, --no_tailreorder* on page 2-131
- *--thumb2_library, --no_thumb2_library* on page 2-132
- *--tiebreaker=option* on page 2-133
- *--undefined=symbol* on page 2-134
- *--undefined_and_export=symbol* on page 2-135
- *--unresolved=symbol* on page 2-136
- *--use_definition_visibility* on page 2-137
- *--userlibpath=pathlist* on page 2-138
- *--veneer_inject_type=type* on page 2-139
- *--veneer_pool_size=size* on page 2-140
- *--veneershare, --no_veneershare* on page 2-141
- *--verbose* on page 2-142
- *--version_number* on page 2-143
- *--vfemode=mode* on page 2-144
- *--via=file* on page 2-145
- *--vsn* on page 2-146
- *--xref, --no_xref* on page 2-147
- *--xrefdbg, --no_xrefdbg* on page 2-148
- *--xref{from|to}=object(section)* on page 2-149
- *--zi_base=address* on page 2-150.

2.1 `--any_contingency`

This option permits extra space in any execution regions containing `.ANY` sections for linker-generated content such as veneers and alignment padding. Two percent of the space is reserved for veneers.

When a region is about to overflow because of potential padding, `armlink` lowers the priority of the `.ANY` selector.

This option is off by default. That is, `armlink` does not attempt to calculate padding and strictly follows the `.ANY` priorities.

Use this option with the `--scatter` option.

2.1.1 See also

Tasks

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

- *Behavior when .ANY sections overflow because of linker-generated content* on page 4-28.

Reference

- *--any_placement=algorithm* on page 2-6
- *--any_sort_order=order* on page 2-8
- *--info=topic[,topic,...]* on page 2-59
- *--scatter=file* on page 2-110
- *Syntax of an input section description* on page 4-22.

2.2 `--any_placement=`*`algorithm`*

Controls the placement of sections that are placed using the `.ANY` module selector.

2.2.1 Syntax

`--any_placement=`*`algorithm`*

where *`algorithm`* is one of the following:

`best_fit` Place the section in the execution region that currently has the least free space but is also sufficient to contain the section.

`first_fit` Place the section in the first execution region that has sufficient space. The execution regions are examined in the order they are defined in the scatter file.

`next_fit` Place the section using the following rules:

- place in the current execution region if there is sufficient free space
- place in the next execution region only if there is insufficient space in the current region
- never place a section in a previous execution region.

`worst_fit` Place the section in the execution region that currently has the most free space.

Use this option with the `--scatter` option.

2.2.2 Usage

The placement algorithms interact with scatter files and `--any_contingency` as follows:

Interaction with normal scatter-loading rules

Scatter-loading with or without `.ANY` assigns a section to the most specific selector. All algorithms continue to assign to the most specific selector in preference to `.ANY` priority or size considerations.

Interaction with `.ANY` priority

Priority is considered after assignment to the most specific selector in all algorithms.

`worst_fit` and `best_fit` consider priority before their individual placement criteria. For example, you might have `.ANY1` and `.ANY2` selectors, with the `.ANY1` region having the most free space. When using `worst_fit` the section is assigned to `.ANY2` because it has higher priority. Only if the priorities are equal does the algorithm come into play.

`first_fit` considers the most specific selector first, then priority. It does not introduce any more placement rules.

`next_fit` also does not introduce any more placement rules. If a region is marked full during `next_fit`, that region cannot be considered again regardless of priority.

Interaction with `--any_contingency`

The priority of a `.ANY` selector is reduced to `0` if the region might overflow because of linker-generated content. This is enabled and disabled independently of the sorting and placement algorithms.

`armlink` calculates a worst-case contingency for each section.

For `worst_fit`, `best_fit`, and `first_fit`, when a region is about to overflow because of the contingency, `armlink` lowers the priority of the related `.ANY` selector.

For `next_fit`, when a possible overflow is detected, `armlink` marks that section as `FULL` and does not consider it again. This stays consistent with the rule that when a section is full it can never be revisited.

2.2.3 Default

The default option is `worst_fit`.

2.2.4 See also

Tasks

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

- *Behavior when .ANY sections overflow because of linker-generated content* on page 4-28

Using the Linker:

- *Examples of using placement algorithms for .ANY sections* on page 8-26
- *Example of next_fit algorithm showing behavior of full regions, selectors, and priority* on page 8-28.

Reference

- *--any_contingency* on page 2-5
- *--any_sort_order=order* on page 2-8
- *--info=topic[,topic,...]* on page 2-59
- *--scatter=file* on page 2-110
- *Syntax of an input section description* on page 4-22.

2.3 `--any_sort_order=`*`order`*

Controls the sort order of input sections that are placed using the `.ANY` module selector.

2.3.1 Syntax

`--any_sort_order=`*`order`*

where *`order`* is one of the following:

`descending_size`	Sort input sections in descending size order.
`cmdline`	Sort input sections by command-line index.

By default, sections that have the same properties are resolved using the creation index. You can use the `--tiebreaker` command-line option to resolve sections by the order they appear on the linker command-line.

Use this option with the `--scatter` option.

2.3.2 Usage

The sorting governs the order that sections are processed during `.ANY` assignment. Normal scatter-loading rules, for example `RO` before `RW`, are obeyed after the sections are assigned to regions.

2.3.3 Default

The default option is `descending_size`.

2.3.4 See also

Tasks

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

Using the Linker:

- *Examples of using sorting algorithms for .ANY sections* on page 8-30.

Reference

- *--any_contingency* on page 2-5
- *--any_placement=algorithm* on page 2-6
- *--info=topic[,topic,...]* on page 2-59
- *--scatter=file* on page 2-110
- *--tiebreaker=option* on page 2-133
- *Syntax of an input section description* on page 4-22.

2.4 `--arm_only`

This option enables the linker to target the ARM instruction set only. If the linker detects any objects requiring Thumb® state, an error is generated.

2.4.1 See also

Reference

Compiler Reference:

- *--arm* on page 3-11
- *--arm_only* on page 3-11
- *--thumb* on page 3-90.

Assembler Reference:

- *--arm* on page 2-6
- *--arm_only* on page 2-6
- *--thumb* on page 2-23.

2.5 `--autoat`, `--no_autoat`

This option controls the automatic assignment of `__at` sections to execution regions. `__at` sections are sections that must be placed at a specific address.

2.5.1 Usage

If enabled, the linker automatically selects an execution region for each `__at` section. If a suitable execution region does not exist, the linker creates a load region and an execution region to contain the `__at` section.

If disabled, the standard scatter-loading section selection rules apply.

2.5.2 Default

The default is `--autoat`.

2.5.3 Restrictions

You cannot use `__at` section placement with position independent execution regions.

2.5.4 See also

Concepts

Using the Linker:

- *Automatic placement of __at sections* on page 8-37
- *Manual placement of __at sections* on page 8-39.

Reference

- Chapter 4 *Formal syntax of the scatter file*.

2.6 `--be8`

This option specifies ARMv6 Byte Invariant Addressing big-endian mode.

This is the default Byte Addressing mode for ARMv6 and later big-endian images.It means that the linker reverses the endianness of the instructions to give little-endian code and big-endian data for input objects that have been compiled or assembled as big-endian.

Byte Invariant Addressing mode is only available on ARM processors that support ARMv6 and above.

2.6.1 See also

Other information

- ARM Architecture Reference Manuals, http://infocenter.arm.com/help/topic/com.arm.doc.subset.arch.reference.

2.7 `--be32`

This option specifies legacy Word Invariant Addressing big-endian mode, that is, identical to big-endian images prior to ARMv6. This produces big-endian code and data.

Word Invariant Addressing mode is the default mode for all pre-ARMv6 big-endian images.

2.7.1 See also

Other information

- ARM Architecture Reference Manuals, http://infocenter.arm.com/help/topic/com.arm.doc.subset.arch.reference/index.html.

2.8 `--bestdebug`, `--no_bestdebug`

This option selects between linking for smallest code/data size or best debug illusion. Input objects might contain common data (COMDAT) groups, but these might not be identical across all input objects because of differences such as objects compiled with different optimization levels.

2.8.1 Default

The default is `--no_bestdebug`. This ensures that the code and data of the final image are the same regardless of whether you compile for debug or not. The smallest COMDAT groups are selected when linking, at the expense of a possibly slightly poorer debug illusion.

2.8.2 Usage

Use `--bestdebug` to select COMDAT groups with the best debug view. Be aware that the code and data of the final image might not be the same when building with or without debug.

2.8.3 Example

For two objects compiled with different optimization levels:

```
armcc -c -O2 file1.c
armcc -c -O0 file2.c
armlink --bestdebug fil1.o file2.o -o image.axf
```

2.8.4 See also

Concepts

Using the Linker:

- *Elimination of common debug sections* on page 5-2
- *Elimination of common groups or sections* on page 5-3
- *Elimination of unused sections* on page 5-4
- *Elimination of unused virtual functions* on page 5-5.

2.9 `--branchnop, --no_branchnop`

This option causes the linker to replace any branch with a relocation that resolves to the next instruction with a `NOP`. This is the default behavior. However, there are cases where you might want to disable the option, for example, when performing verification or pipeline flushes.

2.9.1 Default

The default is `--branchnop`.

Use `--no_branchnop` to disable this behavior.

2.9.2 See also

Concepts

Using the Linker:

- *Handling branches that optimize to a NOP* on page 5-20.

Reference

- *--inline, --no_inline* on page 2-64
- *--tailreorder, --no_tailreorder* on page 2-131.

2.10 `--callgraph`, `--no_callgraph`

This option creates a file containing a static callgraph of functions. The callgraph gives definition and reference information for all functions in the image.

——— **Note** ———

If you use the `--partial` option to create a partially linked object, then no callgraph file is created.

———————————

2.10.1 Usage

The callgraph file:

- is saved in the same directory as the generated image.
- has the same name as the linked image. Use the `--callgraph_file=`*`filename`* option to specify a different callgraph filename.
- has a default output format of HTML. Use the `--callgraph_output=`*`fmt`* option to control the output format.

——— **Note** ———

If the linker is to calculate the function stack usage, any functions defined in the assembler files must have the appropriate:

- `PROC` and `ENDP` directives
- `FRAME PUSH` and `FRAME POP` directives.

———————————

For each function `func` the linker lists the:

- processor state for which the function is compiled (ARM or Thumb)
- set of functions that call `func`
- set of functions that are called by `func`
- number of times the address of `func` is used in the image.

In addition, the callgraph identifies functions that are:

- called through interworking veneers
- defined outside the image
- permitted to remain undefined (weak references)
- called through a *Procedure Linkage Table* (PLT)
- not called but still exist in the image.

The static callgraph also gives information about stack usage. It lists the:

- size of the stack frame used by each function
- maximum size of the stack used by the function over any call sequence, that is, over any acyclic chain of function calls.

If there is a cycle, or if the linker detects a function with no stack size information in the call chain, `+ Unknown` is added to the stack usage. A reason is added to indicate why stack usage is unknown.

The linker reports missing stack frame information if there is no debug frame information for the function.

For indirect functions, the linker cannot reliably determine which function made the indirect call. This might affect how the maximum stack usage is calculated for a call chain. The linker lists all function pointers used in the image.

Use frame directives in assembly language code to describe how your code uses the stack. These directives ensure that debug frame information is present for debuggers to perform stack unwinding or profiling.

2.10.2 Default

The default is `--no_callgraph`.

2.10.3 See also

Reference

- *--callgraph_file=filename* on page 2-17
- *--callgraph_output=fmt* on page 2-18
- *--cgfile=type* on page 2-19
- *--cgsymbol=type* on page 2-20
- *--cgundefined=type* on page 2-21
- Chapter 4 *Formal syntax of the scatter file*.

Assembler Reference:

- *FRAME POP* on page 5-39
- *FRAME PUSH* on page 5-40
- *FUNCTION or PROC* on page 5-47
- *ENDFUNC or ENDP* on page 5-49.

2.11 `--callgraph_file=`*`filename`*

This option controls the output filename of the callgraph.

2.11.1 Syntax

`--callgraph_file=`*`filename`*

where *`filename`* is the callgraph filename.

The default filename is the same as the linked image.

2.11.2 See also

Reference

- *--callgraph, --no_callgraph* on page 2-15
- *--callgraph_output=fmt* on page 2-18
- *--cgfile=type* on page 2-19
- *--cgsymbol=type* on page 2-20
- *--cgundefined=type* on page 2-21
- *--output=file* on page 2-89
- Chapter 4 *Formal syntax of the scatter file*.

2.12 `--callgraph_output=`*`fmt`*

This option controls the output format of the callgraph.

2.12.1 Syntax

`--callgraph_output=`*`fmt`*

Where *`fmt`* can be one of the following:

`html`	Outputs the callgraph in HTML format.
`text`	Outputs the callgraph in plain text format.

2.12.2 Default

The default is `--callgraph_output=html`.

2.12.3 See also

Reference

- *--callgraph, --no_callgraph* on page 2-15
- *--callgraph_file=filename* on page 2-17
- *--cgfile=type* on page 2-19
- *--cgsymbol=type* on page 2-20
- *--cgundefined=type* on page 2-21
- Chapter 4 *Formal syntax of the scatter file*.

2.13 `--cgfile=`*`type`*

This option controls what files are used to obtain the symbols to be included in the callgraph.

2.13.1 Syntax

`--cgfile=`*`type`*

where *`type`* can be one of the following:

`all`	Includes symbols from all files.
`user`	Includes only symbols from user defined objects and libraries.
`system`	Includes only symbols from system libraries.

2.13.2 Default

The default is `--cgfile=all`.

2.13.3 See also

Reference

- *--callgraph, --no_callgraph* on page 2-15
- *--callgraph_file=filename* on page 2-17
- *--callgraph_output=fmt* on page 2-18
- *--cgsymbol=type* on page 2-20
- *--cgundefined=type* on page 2-21
- Chapter 4 *Formal syntax of the scatter file*.

2.14 `--cgsymbol=`*`type`*

This option controls what symbols are included in the callgraph.

2.14.1 Syntax

`--cgsymbol=`*`type`*

Where *`type`* can be one of the following:

`all`	Includes both local and global symbols.
`locals`	Includes only local symbols.
`globals`	Includes only global symbols.

2.14.2 Default

The default is `--cgsymbol=all`.

2.14.3 See also

Reference

- *--callgraph, --no_callgraph* on page 2-15
- *--callgraph_file=filename* on page 2-17
- *--callgraph_output=fmt* on page 2-18
- *--cgfile=type* on page 2-19
- *--cgundefined=type* on page 2-21
- Chapter 4 *Formal syntax of the scatter file*.

2.15 `--cgundefined=type`

This option controls what undefined references are included in the callgraph.

2.15.1 Syntax

`--cgundefined=type`

Where *type* can be one of the following:

`all`	Includes both function entries and calls to undefined weak references.
`entries`	Includes function entries for undefined weak references.
`calls`	Includes calls to undefined weak references.
`none`	Omits all undefined weak references from the output.

2.15.2 Default

The default is `--cgundefined=all`.

2.15.3 See also

Reference

- *--callgraph, --no_callgraph* on page 2-15
- *--callgraph_file=filename* on page 2-17
- *--callgraph_output=fmt* on page 2-18
- *--cgfile=type* on page 2-19
- *--cgsymbol=type* on page 2-20
- Chapter 4 *Formal syntax of the scatter file*.

2.16 `--combreloc`, `--no_combreloc`

This option enables or disables the linker reordering of the dynamic relocations so that a dynamic loader can process them more efficiently. `--combreloc` is the more efficient option.

2.16.1 Default

The default is `--combreloc`.

2.17 `--comment_section`, `--no_comment_section`

This option controls the inclusion of a comment section `.comment` in the final image.

Use `--no_comment_section` to strip the text in the `.comment` section, to help reduce the image size.

——— Note ———

You can also use the `--filtercomment` option to merge comments.

———————

2.17.1 Default

The default is `--comment_section`.

2.17.2 See also

Concepts

- *--filtercomment, --no_filtercomment* on page 2-51

Using the Linker:

- *About merging comment sections* on page 5-23.

2.18 `--compress_debug`, `--no_compress_debug`

This option causes the linker to compress `.debug_*` sections, if it is sensible to do so. This removes some redundancy and reduces debug table size. Using `--compress_debug` can significantly increase the time required to link an image. Debug compression can only be performed on DWARF3 debug data, not DWARF2.

2.18.1 Default

The default is `--no_compress_debug`.

2.18.2 See also

Other information

- *The DWARF Debugging Standard*, http://dwarfstd.org/

2.19 `--cppinit`, `--no_cppinit`

This option enables the linker to use alternative C++ libraries with a different initialization symbol if required.

2.19.1 Syntax

`--cppinit=`*`symbol`*

If `--cppinit=`*`symbol`* is not specified then the default symbol `__cpp_initialize__aeabi_` is assumed.

`--no_cppinit` does not take a *`symbol`* argument.

2.19.2 Effect

The linker adds a non-weak reference to *`symbol`* if any static constructor or destructor sections are detected.

For `--cppinit=__cpp_initialize__aeabi_`, the linker processes R_ARM_TARGET1 relocations as R_ARM_REL32, because this is required by the `__cpp_initialize__aeabi_` function. In all other cases R_ARM_TARGET1 relocations are processed as R_ARM_ABS32.

2.19.3 See also

Concepts

Using ARM C and C++ Libraries and Floating-Point Support:

- *Initialization of the execution environment and execution of the application* on page 2-55
- *C++ initialization, construction and destruction* on page 2-56.

Reference

- *--ref_cpp_init, --no_ref_cpp_init* on page 2-100.

2.20 `--cpu=list`

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

2.20.1 See also

Reference

- *--cpu=name* on page 2-27.

2.21 `--cpu=`*`name`*

This option enables the linker to determine the target ARM processor. It has the same format as the option supported by the compiler.

2.21.1 Syntax

`--cpu=`*`name`*

Where *`name`* is the name of an ARM processor. For details, see the description of `--cpu=`*`name`* compiler option.

2.21.2 Usage

The link phase fails if any of the component object files rely on features that are incompatible with the selected processor. The linker also uses this option to optimize the choice of system libraries and any veneers that need to be generated when building the final image. The default is to select a CPU that is compatible with all of the component object files. That is, to select the most up-to-date architecture among all input objects.

Note

If the `--cpu` option has a built-in *floating-point unit* (FPU) then the linker implies `--fpu=`*`built-in_fpu`*. For example, `--cpu=Cortex-R4F` implies `--fpu=vfpv3_d16`.

2.21.3 See also

Reference

- *--cpu=list* on page 2-26
- *--fpu=list* on page 2-56
- *--fpu=name* on page 2-57.

Compiler Reference:

- *--cpu=list* on page 3-20
- *--cpu=name* on page 3-20
- *--fpu=list* on page 3-43
- *--fpu=name* on page 3-44.

2.22 --crosser_veneershare, --no_crosser_veneershare

Enables or disables veneer sharing across execution regions.

The default is `--crosser_veneershare`, and enables veneer sharing across execution regions.

`--no_crosser_veneershare` prohibits veneer sharing across execution regions.

2.22.1 See also

Reference

- *--veneershare, --no_veneershare* on page 2-141.

2.23 `--datacompressor=`*`opt`*

This option enables you to specify one of the supplied algorithms for RW data compression. If you do not specify a data compression algorithm, the linker chooses the most appropriate one for you automatically. In general, it is not necessary to override this choice.

2.23.1 Syntax

`--datacompressor=`*`opt`*

Where *`opt`* is one of the following:

`on`	Enables RW data compression to minimize ROM size.
`off`	Disables RW data compression.
`list`	Lists the data compressors available to the linker.
`id`	*`id`* is a data compression algorithm:

Table 2-1 Data compressor algorithms

id	Compression algorithm
0	run-length encoding
1	run-length encoding, with LZ77 on small-repeats
2	complex LZ77 compression

Specifying a compressor adds a decompressor to the code area. If the final image does not have compressed data, the decompressor is not added.

2.23.2 Default

The default is `--datacompressor=on`.

2.23.3 See also

Concepts

Using the Linker:

- *Optimization with RW data compression* on page 5-12
- *How the linker chooses a compressor* on page 5-13
- *How compression is applied* on page 5-15
- *Working with RW data compression* on page 5-16.

2.24 `--debug, --no_debug`

This option controls the generation of debug information in the output file. Debug information includes debug input sections and the symbol/string table.

2.24.1 Default

The default is `--debug`.

2.24.2 Usage

Use `--no_debug` to exclude debug information from the output file. The resulting ELF image is smaller, but you cannot debug it at source level. The linker discards any debug input section it finds in the input objects and library members, and does not include the symbol and string table in the image. This only affects the image size as loaded into the debugger. It has no effect on the size of any resulting binary image that is downloaded to the target.

If you are using `--partial` the linker creates a partially-linked object without any debug data.

Note

Do not use `--no_debug` if a `fromelf --fieldoffsets` step is required. If your image is produced without debug information, `fromelf` cannot:

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

2.24.3 See also

Reference

Using the fromelf Image Converter:

- *--fieldoffsets* on page 4-28.

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

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

2.25.1 Syntax

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

Where *`tag`* can be:

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

2.25.2 See also

Reference

- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--errors=file* on page 2-44
- *--remarks* on page 2-102
- *--strict* on page 2-120.

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

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

You can use the `--remarks` option to display these messages.

2.26.1 Syntax

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

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

2.26.2 See also

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--errors=file* on page 2-44
- *--remarks* on page 2-102
- *--strict* on page 2-120.

2.27 `--diag_style=arm|ide|gnu`

This option changes the formatting of warning and error messages.

2.27.1 Default

The default is `--diag_style=arm`.

2.27.2 Usage

`--diag_style=gnu` matches the format reported by the GNU Compiler, `gcc`.

`--diag_style=ide` matches the format reported by Microsoft Visual Studio.

2.27.3 See also

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--errors=file* on page 2-44
- *--remarks* on page 2-102
- *--strict* on page 2-120.

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

This option suppresses all diagnostic messages that have a specific tag.

2.28.1 Syntax

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

Where *tag* can be:

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

2.28.2 Example

To suppress the warning messages that have numbers `L6314W` and `L6305W`, use the following command:

```
armlink --diag_suppress=L6314,L6305 ...
```

2.28.3 See also

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--errors=file* on page 2-44
- *--remarks* on page 2-102
- *--strict* on page 2-120.

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

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

2.29.1 Syntax

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

Where *`tag`* can be:

- a diagnostic message number to set to warning severity
- `error`, to set all errors that can be downgraded to warnings.

2.29.2 See also

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--errors=file* on page 2-44
- *--remarks* on page 2-102
- *--strict* on page 2-120.

2.30 `--eager_load_debug`, `--no_eager_load_debug`

The `--no_eager_load_debug` option causes the linker to remove debug section data from memory after object loading. This lowers the peak memory usage of the linker at the expense of some linker performance, because much of the debug data has to be loaded again when the final image is written.

Using `--no_eager_load_debug` option does not affect the debug data that is written into the ELF file.

The default is `--eager_load_debug`.

Note

The resulting image or object built without debug information might differ by a small number of bytes. This is because the `.comment` section contains the linker command line used, where the options have differed from the default (the default is `--eager_debug_load`). Therefore `--no_eager_load_debug` images are a little larger and contain Program Header and possibly a Section Header a small number of bytes later. Use `--no_comment_section` to eliminate this difference.

2.30.1 See also

Reference

- *--comment_section, --no_comment_section* on page 2-23.

2.31 `--edit=`*`file_list`*

This option enables you to specify steering files containing commands to edit the symbol tables in the output binary. You can specify commands in a steering file to:

- Hide global symbols. Use this option to hide specific global symbols in object files. The hidden symbols are not publicly visible.
- Rename global symbols. Use this option to resolve symbol naming conflicts.

2.31.1 Syntax

```
--edit=file_list
```

Where *`file_list`* can be more than one steering file separated by a comma. Do not include a space after the comma.

2.31.2 Example

```
--edit=file1 --edit=file2 --edit=file3
```

```
--edit=file1,file2,file3
```

2.31.3 See also

Concepts

Using the Linker:

- *Hiding and renaming global symbols with a steering file* on page 7-27.

Reference

- Chapter 3 *Linker steering file command reference*.

2.32 `--emit_debug_overlay_relocs`

Outputs only relocations of debug sections with respect to overlaid program sections to aid an overlay-aware debugger.

2.32.1 See also

Reference

- *--emit_debug_overlay_section* on page 2-39
- *--emit_non_debug_relocs* on page 2-40
- *--emit_relocs* on page 2-41.

Other information

- *ABI for the ARM Architecture: Support for Debugging Overlaid Programs*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0049-/index.html.

2.33 `--emit_debug_overlay_section`

In a relocatable file, a debug section refers to a location in a program section by way of a relocated location. A reference from a debug section to a location in a program section has the following format:

```
<debug_section_index, debug_section_offset>, <program_section_index, program_section_offset>
```

During static linking the pair of *`program`* values is reduced to single value, the execution address. This is ambiguous in the presence of overlaid sections.

To resolve this ambiguity, use this option to output a `.ARM.debug_overlay` section of type `SHT_ARM_DEBUG_OVERLAY = SHT_LOUSER + 4` containing a table of entries as follows:

```
debug_section_offset, debug_section_index, program_section_index
```

2.33.1 See also

Reference

- *--emit_debug_overlay_relocs* on page 2-38
- *--emit_relocs* on page 2-41.

Other information

- *ABI for the ARM Architecture: Support for Debugging Overlaid Programs*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0049-/index.html.

2.34 `--emit_non_debug_relocs`

Retains only relocations from non-debug sections in an executable file.

2.34.1 See also

Reference

- *--emit_relocs* on page 2-41.

2.35 `--emit_relocs`

Retains all relocations in the executable file. This results in larger executable files.

This is equivalent to the GNU ld `--emit-relocs` option.

2.35.1 See also

Reference

- *--emit_debug_overlay_relocs* on page 2-38
- *--emit_non_debug_relocs* on page 2-40.

Other information

- *ABI for the ARM Architecture: Support for Debugging Overlaid Programs*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0049-/index.html.

2.36 `--entry=location`

This option specifies the unique initial entry point of the image.

2.36.1 Syntax

`--entry=location`

Where *`location`* is one of the following:

`entry_address`

A numerical value, for example: `--entry=0x0`

`symbol` Specifies an image entry point as the address of *`symbol`*, for example: `--entry=reset_handler`

`offset+object(section)`

Specifies an image entry point as an *`offset`* inside a *`section`* within a particular *`object`*, for example:`--entry=8+startup.o(startupseg)`

There must be no spaces within the argument to `--entry`. The input section and object names are matched without case-sensitivity. You can use the following simplified notation:

- `object(section)`, if `offset` is zero.
- `object`, if there is only one input section. `armlink` generates an error message if there is more than one code input section in `object`.

Note

If the entry address of your image is in Thumb state, then the least significant bit of the address must be set to 1. The linker does this automatically if you specify a symbol. For example, if the entry code starts at address `0x8000` in Thumb state you must use `--entry=0x8001`.

Note

If you use `--ltcg`, then only `--entry=symbol` can be used.

2.36.2 Usage

The image can contain multiple entry points, but the initial entry point specified with this option is stored in the executable file header for use by the loader. There can be only one occurrence of this option on the command line. A debugger typically uses this entry address to initialize the *Program Counter* (PC) when an image is loaded. The initial entry point must meet the following conditions:

- the image entry point must lie within an execution region
- the execution region must be non-overlay, and must be a root execution region (load address == execution address).

2.36.3 See also

Concepts

Using the Linker:

- *About link-time code generation* on page 5-10.

Reference

- *--ltcg* on page 2-81
- *--startup=symbol, --no_startup* on page 2-119.

2.37 `--errors=`*`file`*

This option redirects the diagnostics from the standard error stream to *`file`*.

The specified file is created at the start of the link stage. If a file of the same name already exists, it is overwritten.

If *`file`* is specified without path information, it is created in the current directory.

2.37.1 See also

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_style=arm|ide|gnu* on page 2-33
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35.

2.38 `--exceptions`, `--no_exceptions`

This option controls the generation of exception tables in the final image.

2.38.1 Default

`The default is --exceptions.`

2.38.2 Usage

Using `--no_exceptions` generates an error message if any exceptions sections are present in the image after unused sections have been eliminated. Use this option to ensure that your code is exceptions free.

2.38.3 See also

Concepts

Using the Linker:

- *Using command-line options to control the generation of C++ exception tables* on page 4-31.

2.39 `--exceptions_tables=`*`action`*

This option specifies how exception tables are generated for objects that do not already contain exception unwinding tables.

2.39.1 Syntax

`--exceptions_tables=`*`action`*

Where *`action`* is one of the following:

`nocreate`	The linker does not create missing exception tables.
`unwind`	The linker creates an unwinding table for each section in your image that does not already have an exception table.
`cantunwind`	The linker creates a nounwind table for each section in your image that does not already have an exception table.

2.39.2 Default

`The default is --exceptions_tables=nocreate.`

2.39.3 See also

Concepts

Using the Linker:

- *Using command-line options to control the generation of C++ exception tables* on page 4-31.

2.40 `--export_dynamic`, `--no_export_dynamic`

If an executable has dynamic symbols, then `--export_dynamic` exports all externally visible symbols.

2.40.1 Usage

`--export_dynamic` exports non-hidden symbols into the dynamic symbol table only if a dynamic symbol table already exists.

You can use `--export_dynamic` to produce a statically linked image if there are no imports or exports.

`--no_export_dynamic` is the default.

2.41 `--feedback=`*`file`*

This option generates a feedback file for input to the compiler. This file informs the compiler about unused functions.

During your next compilation, use the compiler option `--feedback=`*`file`* to specify the feedback file to use. Unused functions are then placed in their own sections for possible future elimination by the linker.

2.41.1 See also

Concepts

Using the Linker:

- *About linker feedback* on page 5-6.

Reference

- *--feedback_image=option* on page 2-49
- *--feedback_type=type* on page 2-50.

Compiler Reference:

- *--feedback=filename* on page 3-39.

2.42 `--feedback_image=`*`option`*

This option changes the behavior of the linker when writing a feedback file with scatter-loading. Use this option to produce a feedback file where an executable ELF image cannot be created. That is, when your code does not fit into the region limits described in your scatter file before unused functions are removed using linker feedback.

2.42.1 Syntax

`--feedback_image=`*`option`*

Where *`option`* is one of the following:

`none`	Uses the scatter file to determine region size limits. Disables region overlap and region size overflow messages. Does not write an ELF image. Error messages are still produced if a region overflows the 32-bit address space.
`noerrors`	Uses the scatter file to determine region size limits. Warns on region overlap and region size overflow messages. Writes an ELF image, which might not be executable. Error messages are still produced if a region overflows the 32-bit address space.
`simple`	Ignores the scatter file. Disables ROPI/RWPI errors and warnings. Writes an ELF image, which might not be executable.
`full`	Enables all error and warning messages and writes a valid ELF image.

2.42.2 Default

The default option is `--feedback_image=full`.

2.42.3 See also

Concepts

Using the Linker:

- *About linker feedback* on page 5-6.

Reference

- *--feedback=file* on page 2-48
- *--feedback_type=type* on page 2-50
- *--scatter=file* on page 2-110.

Compiler Reference:

- *--feedback=filename* on page 3-39.

2.43 `--feedback_type=type`

This option controls the information that the linker puts into the feedback file.

2.43.1 Syntax

`--feedback_type=type`

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

`[no]iw`	controls functions that require interworking support.
`[no]unused`	controls unused functions in the image.

2.43.2 Default

The default option is `--feedback_type=unused,noiw`.

2.43.3 See also

Concepts

Using the Linker:

- *About linker feedback* on page 5-6.

Reference

- *--feedback=file* on page 2-48
- *--feedback_image=option* on page 2-49.

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-7
- *--feedback=filename* on page 3-39.

2.44 `--filtercomment`, `--no_filtercomment`

The linker always removes identical comments. The `--filtercomment` permits the linker to pre-process the `.comment` section and remove some information that prevents merging.

Use `--no_filtercomment` to prevent the linker from modifying the `.comment` section.

2.44.1 Default

The default is `--filtercomment`.

2.44.2 See also

Concepts

Using the Linker:

- *About merging comment sections* on page 5-23.

2.45 `--fini=`*`symbol`*

This option specifies the symbol name that is used to define the entry point for finalization code. The dynamic linker executes this code when it unloads the executable file or shared object.

2.45.1 See also

Reference

- *--init=symbol* on page 2-63
- *--symbolic* on page 2-128.

2.46 `--first=section_id`

This option places the selected input section first in its execution region. This can, for example, place the section containing the vector table first in the image.

2.46.1 Syntax

`--first=section_id`

Where `section_id` is one of the following:

`symbol` Selects the section that defines `symbol`. You must not specify a symbol that has more than one definition, because only one section can be placed first. For example: `--first=reset`

`object(section)`

Selects `section` from `object`. There must be no space between `object` and the following open parenthesis. For example: `--first=init.o(init)`

`object` Selects the single input section in `object`. If you use this short form and there is more than one input section, the linker generates an error message. For example: `--first=init.o`

2.46.2 Usage

The `--first` option cannot be used with `--scatter`. Instead, use the `+FIRST` attribute in a scatter file.

2.46.3 See also

Concepts

Using the Linker:

- *Section placement with the linker* on page 4-19
- *Placing sections with FIRST and LAST attributes* on page 4-21.

Reference

- *--last=section_id* on page 2-72
- *--scatter=file* on page 2-110.

2.47 `--force_explicit_attr`

The `--cpu` option checks the FPU attributes if the CPU chosen has a built-in FPU.

The error message `L6463E: Input Objects contain` *`archtype`* `instructions but could not find valid target for` *`archtype`* `architecture based on object attributes. Suggest using --cpu option to select a specific cpu.` is given in one of two situations:

- the ELF file contains instructions from architecture *`archtype`* yet the build attributes claim that *`archtype`* is not supported
- the build attributes are inconsistent enough that the linker cannot map them to an existing CPU.

If setting the `--cpu` option still fails, use `--force_explicit_attr` to cause the linker to retry the CPU mapping using build attributes constructed from `--cpu=`*`archtype`*. This might help if the error is being given solely because of inconsistent build attributes.

2.47.1 See also

Reference

- *--cpu=name* on page 2-27
- *--fpu=name* on page 2-57.

Compiler Reference:

- *--cpu=name* on page 3-20
- *--fpu=name* on page 3-44.

Assembler Reference:

- *--cpu=name* on page 2-8
- *--fpu=name* on page 2-13.

2.48 `--force_so_throw`, `--no_force_so_throw`

This option controls the assumption made by the linker that an input shared object might throw an exception. By default, exception tables are discarded if no code throws an exception.

2.48.1 Default

`The default is --no_force_so_throw.`

2.48.2 Usage

Use `--force_so_throw` to specify that all shared objects might throw an exception and so force the linker to keep the exception tables, regardless of whether the image can throw an exception or not.

2.49 `--fpu=list`

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

2.49.1 See also

Reference

- *--fpu=name* on page 2-57.

2.50 `--fpu=`*`name`*

This option enables the linker to determine the target FPU architecture.

The linker fails if any of the component object files rely on features that are incompatible with the selected FPU architecture. The linker also uses this option to optimize the choice of system libraries. The default is to select an FPU that is compatible with all of the component object files.

This option has the same format as that supported by the compiler.

2.50.1 See also

Reference

- *--fpu=list* on page 2-56.

Compiler Reference:

- *--cpu=name* on page 3-20
- *--fpu=list* on page 3-43
- *--fpu=name* on page 3-44.

2.51 `--help`

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

2.51.1 Default

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

2.51.2 See also

Reference

- *--show_cmdline* on page 2-112
- *--version_number* on page 2-143
- *--vsn* on page 2-146.

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

This option prints information about specific topics. You can write the output to a text file using `--list=file`.

2.52.1 Syntax

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

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

`any` For sections placed using the `.ANY` module selector, lists:

- the sort order
- the placement algorithm
- the sections that are assigned to each execution region in the order they are assigned by the placement algorithm.
- Information about the continency space and policy used for each region.

This keyword also displays additional information when you use the execution region attribute `ANY_SIZE` in a scatter file.

`architecture` Summarizes the image architecture by listing the CPU, FPU and byte order.

`common` Lists all common sections that are eliminated from the image. Using this option implies `--info=common,totals`.

`compression` Gives extra information about the RW compression process.

`debug` Lists all rejected input debug sections that are eliminated from the image as a result of using `--remove`. Using this option implies `--info=debug,totals`.

`exceptions` Gives information on exception table generation and optimization.

`inline` Lists all functions that are inlined by the linker, and the total number of inlines if `--inline` is used.

`inputs` Lists the input symbols, objects and libraries.

`libraries` Lists the full path name of every library automatically selected for the link stage.

You can use this option with `--info_lib_prefix` to display information about a specific library.

`merge` Lists the **`const`** strings that are merged by the linker. Each item lists the merged result, the strings being merged, and the associated object files.

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

`stack` Lists the stack usage of all global symbols.

`summarysizes` Summarizes the code and data sizes of the image.

`summarystack` Summarizes the stack usage of all global symbols.

`tailreorder` Lists all the tail calling sections that are moved above their targets, as a result of using `--tailreorder`.

`totals` Lists the totals of the code and data (RO Data, RW Data, ZI Data, and Debug Data) sizes for input objects and libraries.

`unused`	Lists all unused sections that are eliminated from the user code as a result of using `--remove`. It does not list any unused sections that are loaded from the ARM C libraries.
`unusedsymbols`	Lists all symbols that have been removed by unused section elimination.
`veneers`	Lists the linker-generated veneers.
`veneercallers`	Lists the linker-generated veneers with additional information about the callers to each veneer. Use with `--verbose` to list each call individually.
`veneerpools`	Displays information on how the linker has placed veneer pools.
`visibility`	Lists the symbol visibility information. You can use this option with either `--info=inputs` or `--verbose` to enhance the output.
`weakrefs`	Lists all symbols that are the target of weak references, and whether or not they were defined.

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

If you are using RW data compression (the default), or if you have specified a compressor using the `--datacompressor=`*`id`* option, the output from `--info=sizes,totals` includes an entry under `Grand Totals` to reflect the true size of the image.

——— **Note** ———

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

———————————

2.52.2 See also

Tasks

Using the Linker:

- *Linker options for getting information about images* on page 6-2
- *Working with RW data compression* on page 5-16
- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

Using the Linker:

- *Elimination of unused sections* on page 5-4
- *Optimization with RW data compression* on page 5-12
- *How the linker chooses a compressor* on page 5-13
- *How compression is applied* on page 5-15.

Reference

- *--any_placement=algorithm* on page 2-6
- *--any_sort_order=order* on page 2-8
- *--datacompressor=opt* on page 2-29
- *--info_lib_prefix=opt* on page 2-62
- *--inline, --no_inline* on page 2-64
- *--merge, --no_merge* on page 2-87

- *--remove, --no_remove* on page 2-103
- *--tailreorder, --no_tailreorder* on page 2-131
- *--veneer_inject_type=type* on page 2-139
- *--verbose* on page 2-142
- *Execution region attributes* on page 4-11.

2.53 `--info_lib_prefix=opt`

This option is a filter for the `--info=libraries` option. The linker only displays the libraries that have the same prefix as the filter.

2.53.1 Syntax

```
armlink --info=libraries --info_lib_prefix=opt
```

Where *opt* is the prefix of the required library.

2.53.2 Example

- Displaying a list of libraries without the filter:

  ```
  armlink --info=libraries test.o
  ```

  Produces a list of libraries, for example:

  ```
  install_directory\RV31\LIB\armlib\c_4.l
  install_directory\RV31\LIB\armlib\fz_4s.l install_directory\RV31\LIB\armlib\h_4.l
  install_directory\RV31\LIB\armlib\m_4s.l
  install_directory\RV31\LIB\armlib\vfpsupport.l
  ```

- Displaying a list of libraries with the filter:

  ```
  armlink --info=libraries --info_lib_prefix=c test.o
  ```

  Produces a list of libraries with the specified prefix, for example:

  ```
  install_directory\RV31\LIB\armlib\c_4.l
  ```

2.53.3 See also

Reference

- *--info=topic[,topic,...]* on page 2-59.

2.54 `--init=`*`symbol`*

This option specifies the symbol name that is used to define initialization code. A dynamic linker executes this code when it loads the executable file or shared object.

2.54.1 See also

Reference

- *--fini=symbol* on page 2-52
- *--symbolic* on page 2-128.

2.55 `--inline`, `--no_inline`

This option enables or disables branch inlining to optimize small function calls in your image.

2.55.1 Default

The default is `--no_inline`.

Note

This branch optimization is off by default because enabling it changes the image such that debug information might be incorrect. If enabled, the linker makes no attempt to correct the debug information.

2.55.2 See also

Tasks

Using the Linker:

- *Inlining functions with the linker* on page 5-17.

Reference

- *--branchnop, --no_branchnop* on page 2-14
- *--tailreorder, --no_tailreorder* on page 2-131.

2.56 `--inlineveneer, --no_inlineveneer`

This option enables or disables the generation of inline veneers to give greater control over how the linker places sections.

2.56.1 Default

The default is `--inlineveneer`.

2.56.2 See also

Concepts

Using the Linker:

- *Overview of veneers* on page 4-26
- *Veneer sharing* on page 4-27
- *Veneer types* on page 4-28
- *Generation of position independent to absolute veneers* on page 4-29
- *Reuse of veneers when scatter-loading* on page 4-30.

Reference

- *--piveneer, --no_piveneer* on page 2-95
- *--veneershare, --no_veneershare* on page 2-141.

2.57 `input-file-list`

This is a space-separated list of objects, libraries, or *symbol definitions* (symdefs) files.

2.57.1 Usage

The linker sorts through the input file list in order. If the linker is unable to resolve input file problems then a diagnostic message is produced.

The symdefs files can be included in the list to provide global symbol addresses for previously generated image files.

You can use libraries in the input file list in the following ways:

- Specify a library to be added to the list of libraries that is used to extract members if they resolve any non weak unresolved references. For example, specify `mystring.lib` in the input file list.

  Note

  Members from the libraries in this list are added to the image only when they resolve an unresolved non weak reference.

- Specify particular members to be extracted from a library and added to the image as individual objects. Members are selected from a comma separated list of patterns that can include wild characters. Spaces are permitted but if you use them you must enclose the whole input file list in quotes.

  The following shows an example of an input file list both with and without spaces:

  ```
  mystring.lib(strcmp.o,std*.o)
  “mystring.lib(strcmp.o, std*.o)”
  ```

The linker automatically searches the appropriate C and C++ libraries in order to select the best standard functions for your image. You can use `--no_scanlib` to prevent automatic searching of the standard system libraries.

The linker processes the input file list in the following order:

1. Objects are added to the image unconditionally.
2. Members selected from libraries using patterns are added to the image unconditionally, as if they are objects. For example, to add all `a*.o` objects and `stdio.o` from `mystring.lib` use the following:

   ```
   “mystring.lib(stdio.o, a*.o)”
   ```

3. Library files listed on the command-line are searched for any unresolved non-weak references. The standard C or C++ libraries are added to the list of libraries that are later used to resolve any remaining references.

2.57.2 See also

Tasks

Using the Linker:

- *Accessing symbols in another image* on page 7-17.

Concepts

Using the Linker:

- *How the linker performs library searching, selection, and scanning* on page 4-35.

Reference

- *--scanlib, --no_scanlib* on page 2-109.

2.58 `--keep=section_id`

This option specifies input sections that must not be removed by unused section elimination.

2.58.1 Syntax

`--keep=section_id`

Where *`section_id`* is one of the following:

`symbol` Specifies that an input section defining *`symbol`* is to be retained during unused section elimination. If multiple definitions of *`symbol`* exist, `armlink` generates an error message.

For example, you might use `--keep=int_handler`.

To keep all sections that define a symbol ending in `_handler`, use `--keep=*_handler`.

`object``(`*`section`*`)`

Specifies that *`section`* from *`object`* is to be retained during unused section elimination. If a single instance of *`section`* is generated, you can omit *`section`*, for example, `file.o()`. Otherwise, you must specify *`section`*.

For example, to keep the `vect` section from the `vectors.o` object use: `--keep=vectors.o(vect)`

To keep all sections from the `vectors.o` object where the first three characters of the name of the sections are `vec`, use:`--keep=vectors.o(vec*)`

`object` Specifies that the single input section from *`object`* is to be retained during unused section elimination. If you use this short form and there is more than one input section in *`object`*, the linker generates an error message.

For example, you might use `--keep=dspdata.o`.

To keep the single input section from each of the objects that has a name starting with `dsp`, use `--keep=dsp*.o`.

All forms of the *`section_id`* argument can contain the `*` and `?` wild characters. Matching is case-insensitive, even on hosts with case-sensitive file naming. For example:

- `--keep foo.o(Premier*)` causes the entire match for `Premier*` to be case-insensitive
- `--keep foo.o(Premier)` causes a case-sensitive match for the string Premier.

Use `*.o` to match all object files. Use `*` to match all object files and libraries.

You can specify multiple `--keep` options on the command line.

2.58.2 Matching a symbol that has the same name as an object

If you name a symbol with the same name as an object, then `--keep=`*`symbol_id`* searches for a symbol that matches *`symbol_id`*:

- If a symbol is found, it matches the symbol.
- If no symbol is found, it matches the object.

You can force `--keep` to match an object with `--keep=`*`symbol_id`*`()`. Therefore, to keep both the symbol and the object, specify `--keep foo.o --keep foo.o()`.

2.58.3 See also

Concepts

Using the Linker:

- *The image structure* on page 4-3.

2.59 `--keep_protected_symbols`

Use this option to explicitly keep STV_PROTECTED symbols even if you are not using dynamic linking.

For example, your application might export functions provided by an API to shared objects that are loaded using a custom loader. However, the linker unused section elimination optimization causes the sections to be removed, even if those sections include STV_PROTECTED symbols. To prevent section containing STV_PROTECTED symbols from being removed, specify `--keep_protected_symbols`.

2.59.1 See also

Concepts

Using the Linker:

- *Elimination of unused sections* on page 5-4.

Reference

- *--max_visibility=type* on page 2-86
- *--override_visibility* on page 2-90

2.60 --largeregions, --no_largeregions

This option controls the sorting order of sections in large execution regions to minimize the distance between sections that call each other.

2.60.1 Usage

If the execution region contains more code than the range of a branch instruction then the linker switches to large region mode. In this mode the linker sorts according to the approximated average call depth of each section in ascending order. The linker might also place distribute veneers amongst the code sections to minimize the number of veneers.

Note

Large region mode can result in large changes to the layout of an image even when small changes are made to the input.

To disable large region mode and revert to lexical order, use `--no_largeregions`. Section placement is then predictable and image comparisons are more predictable. However some branches might not reach the target causing the link step to fail. If this happens you must place code/data sections explicitly using an appropriate scatter file or write your own veneer.

2.60.2 Default

The default is `--no_largeregions`. The linker automatically switches to `--largeregions` if at least one execution region contains more code than the smallest inter-section branch. The smallest inter-section branch depends on the code in the region and the target processor:

32Mb	Execution region contains only ARM.
16Mb	Execution region contains Thumb, Thumb-2 is supported.
4Mb	Execution region contains Thumb, no Thumb-2 support.

2.60.3 See also

Concepts

Using the Linker:

- *Overview of veneers* on page 4-26
- *Veneer sharing* on page 4-27
- *Veneer types* on page 4-28
- *Generation of position independent to absolute veneers* on page 4-29
- *Reuse of veneers when scatter-loading* on page 4-30.

Reference

- *--sort=algorithm* on page 2-116.

2.61 `--last=section_id`

This option places the selected input section last in its execution region. For example, this can force an input section that contains a checksum to be placed last in the RW section.

2.61.1 Syntax

`--last=section_id`

Where `section_id` is one of the following:

`symbol`	Selects the section that defines `symbol`. You must not specify a symbol that has more than one definition because only a single section can be placed last. For example: `--last=checksum`
`object(section)`	Selects the `section` from `object`. There must be no space between `object` and the following open parenthesis. For example: `--last=checksum.o(check)`
`object`	Selects the single input section from `object`. If there is more than one input section in `object`, `armlink` generates an error message.

2.61.2 Usage

The `--last` option cannot be used with `--scatter`. Instead, use the `+LAST` attribute in a scatter file.

2.61.3 See also

Concepts

Using the Linker:

- *Section placement with the linker* on page 4-19
- *Placing sections with FIRST and LAST attributes* on page 4-21.

Reference

- *--first=section_id* on page 2-53
- *--scatter=file* on page 2-110.

2.62 `--ldpartial`

This option enables you to link a partial object with the linker combining sections in the output object. This contrasts with the `--partial` option which does not combine sections. You can control the section combination with a scatter file.

`-r` is a synonym for `--ldpartial`.

2.63 `--legacyalign`, `--no_legacyalign`

By default, the linker assumes execution regions and load regions to be four-byte aligned. This option enables the linker to minimize the amount of padding that it inserts into the image.

The `--no_legacyalign` option instructs the linker to insert padding to force natural alignment. Natural alignment is the highest known alignment for that region.

Use `--no_legacyalign` to ensure strict conformance with the ELF specification.

You can also use expression evaluation in a scatter file to avoid padding.

2.63.1 See also

Concepts

Using the Linker:

- *Section placement with the linker* on page 4-19.

Reference

- *Load region attributes* on page 4-7
- *Execution region attributes* on page 4-11
- *Using expression evaluation in a scatter file to avoid padding* on page 8-57.

2.64 `--libpath=`*`pathlist`*

This option specifies a list of paths that are used to search for the ARM standard C and C++ libraries.

You can also use the `ARMCC`*`nn`*`LIB` environment variable to specify the path for the parent directory containing the ARM libraries is specified by , where *`nn`* is the version of the compilation tools installed. For example.`ARMCC41LIB`. Any paths specified with `--libpath` override the path specified by the environment variable.

2.64.1 Syntax

`--libpath=`*`pathlist`*

Where *`pathlist`* is a comma-separated list of paths that are only used to search for required ARM libraries. Do not include spaces between the comma and the path name when specifying multiple path names, for example, *`path1,path2,path3,...,pathn`*.

Note

This option does not affect searches for user libraries. Use `--userlibpath` instead for user libraries.

2.64.2 See also

Concepts

Using the Linker:

- *How the linker performs library searching, selection, and scanning* on page 4-35.

Reference

- *--userlibpath=pathlist* on page 2-138.

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-12

2.65 `--library_type=`*`lib`*

This option selects the library to be used at link time.

Note

This option can be used with the compiler, assembler or linker.

Use this option with the linker to override all other `--library_type` options.

2.65.1 Syntax

`--library_type=`*`lib`*

Where *`lib`* can be one of:

`standardlib`	Specifies that the full runtime libraries are selected at link time.
`microlib`	Specifies that the *C micro-library* (microlib) is selected at link time.

2.65.2 Default

If you do not specify `--library_type` at link time and no object file specifies a preference, then the linker assumes `--library_type=standardlib`.

2.65.3 See also

Concepts

Using the ARM® C and C++ Libraries and Floating Point Support:

- *Building an application with microlib* on page 3-7.

2.66 `--list=`*`file`*

This option redirects the diagnostics output by the `--info`, `--map`, `--symbols`, `--verbose`, `--xref`, `--xreffrom`, and `--xrefto` options to *`file`*.

The specified file is created when diagnostics are output. If a file of the same name already exists, it is overwritten. However, if diagnostics are not output, a file is not created. In this case, the contents of any existing file with the same name remain unchanged.

If *`file`* is specified without a path, it is created in the output directory, that is, the directory where the output image is being written.

2.66.1 See also

Reference

- *--info=topic[,topic,...]* on page 2-59
- *--map, --no_map* on page 2-83
- *--symbols, --no_symbols* on page 2-129
- *--verbose* on page 2-142
- *--xref, --no_xref* on page 2-147
- *--xrefdbg, --no_xrefdbg* on page 2-148.

2.67 `--list_mapping_symbols`, `--no_list_mapping_symbols`

This option enables or disables the addition of mapping symbols `$a`, `$d`, `$t`, and `$t.x` in the output produced by `--symbols`.

Mapping symbols are used to flag transitions between ARM code, Thumb code, and data.

2.67.1 Default

The default is `--no_list_mapping_symbols`.

2.67.2 See also

Concepts

Using the Linker:

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

Reference

- *--symbols, --no_symbols* on page 2-129.

Other information

- *ELF for the ARM Architecture*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044-/index.html.

2.68 `--load_addr_map_info`, `--no_load_addr_map_info`

This option includes load addresses for execution regions in the map file.

If an input section is compressed, then the load address has no meaning and `COMPRESSED` is displayed instead.

For sections that do not have a load address, such as ZI data, the load address is blank

2.68.1 Default

The default is `--no_load_addr_map_info`.

2.68.2 Restrictions

You must use the `--map` with this option.

2.68.3 Example

The following example shows the format of the map file output:

```
Base Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

0x00008000   0x00008000   0x00000008   Code   RO          25   * !!!main             __main.o(c_4.l)
0x00010000   COMPRESSED   0x00001000   Data   RW           2     dataA               data.o
0x00003000        -       0x00000004   Zero   RW           2     .bss                test.o
```

2.68.4 See also

Reference

- *--map, --no_map* on page 2-83.

2.69 `--locals`, `--no_locals`

The `--locals` option adds local symbols in the output symbol table.

The effect of the `--no_locals` option is different for images and object files.

When producing an executable image `--no_locals` removes local symbols from the output symbol table.

For object files built with the `--partial` option, the `--no_locals` 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.

`--no_locals` is a useful optimization if you want to reduce the size of the output symbol table in the final image.

2.69.1 Default

The default is `--locals`.

2.69.2 See also

Reference

- *--privacy* on page 2-98.

2.70 `--ltcg`

This option enables *link-time code generation* (LTCG). You must use this option if any of your input objects have been compiled with `--ltcg`.

——— Note ———

The LTCG feature is deprecated. As an alternative ARM recommends you use the `--multifile` compiler option.

2.70.1 See also

Concepts

Using the Linker:

- *About link-time code generation* on page 5-10.

Reference

Compiler Reference:

- *--ltcg* on page 3-64
- *--multifile, --no_multifile* on page 3-67.

2.71 `--mangled`, `--unmangled`

This option instructs the linker to display mangled or unmangled C++ symbol names in diagnostic messages, and in listings produced by the `--xref`, `--xreffrom`, `--xrefto`, and `--symbols` options.

2.71.1 Default

The default is `--unmangled`.

2.71.2 Usage

If `--unmangled` is selected, C++ symbol names are displayed as they appear in your source code.

If `--mangled` is selected, C++ symbol names are displayed as they appear in the object symbol tables.

2.71.3 See also

Reference

- *--match=crossmangled* on page 2-84
- *--symbols, --no_symbols* on page 2-129
- *--xref, --no_xref* on page 2-147
- *--xrefdbg, --no_xrefdbg* on page 2-148
- *--xref{from|to}=object(section)* on page 2-149.

2.72 `--map, --no_map`

This option enables or disables the printing of a memory map.

The map contains the address and the size of each load region, execution region, and input section in the image, including linker-generated input sections. This can be output to a text file using `--list=`*`file`*.

2.72.1 Default

The default is `--no_map`.

2.72.2 See also

Tasks

Using the Linker:

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

Reference

- *--list=file* on page 2-77
- *--load_addr_map_info, --no_load_addr_map_info* on page 2-79
- *--section_index_display=type* on page 2-111.

2.73 `--match=crossmangled`

This option instructs the linker to match the following combinations together:

- a reference to an unmangled symbol with the mangled definition
- a reference to a mangled symbol with the unmangled definition.

Libraries and matching combinations operate as follows:

- If the library members define a mangled definition, and there is an unresolved unmangled reference, the member is loaded to satisfy it.
- If the library members define an unmangled definition, and there is an unresolved mangled reference, the member is loaded to satisfy it.

——— **Note** ———

This option has no effect if used with partial linking. The partial object contains all the unresolved references to unmangled symbols, even if the mangled definition exists. Matching is done only in the final link step.

———————————

2.73.1 See also

Reference

- *--mangled, --unmangled* on page 2-82.

2.74 `--max_veneer_passess=`*`value`*

This option specifies a limit to the number of veneer generation passes the linker attempts to make when both the following conditions are met:

- a Section that is sufficiently large has a relocation that requires a veneer
- the linker cannot place the veneer close enough to the call site.

The linker attempts to diagnose the failure if the maximum number of veneer generation passes you specify is exceeded, and displays a warning message. You can downgrade this warning message using `--diag_remark`.

2.74.1 Syntax

`--max_veneer_passes=`*`value`*

Where *`value`* is the maximum number of veneer passes the linker is to attempt. The minimum value you can specify is one.

2.74.2 Default

The default number of passes is 10.

2.74.3 See also

Reference

- *--diag_remark=tag[,tag,...]* on page 2-32
- *--diag_warning=tag[,tag,...]* on page 2-35.

2.75 `--max_visibility=type`

This option controls the visibility of all symbol definitions.

2.75.1 Syntax

`--max_visibility=type`

Where *type* can be one of:

`default`	Default visibility.
`protected`	Protected visibility.

2.75.2 Usage

Use`--max_visibility=protected` to limit the visibility of all symbol definitions. Global symbol definitions that normally have default visibility, are given protected visibility when this option is specified.

2.75.3 Default

The default is `--max_visibility=default`.

2.75.4 See also

Reference

- *--keep_protected_symbols* on page 2-70
- *--override_visibility* on page 2-90.

2.76 `--merge`, `--no_merge`

This option enables or disables the merging of **`const`** strings that are placed in shareable sections by the compiler. Using `--merge` can reduce the size of the image if there are similarities between **`const`** strings.

For a listing of the merged **`const`** strings you can use `--info=merge`.

2.76.1 Default

The default is `--merge`.

By default, merging happens between different load and execution regions. Therefore, code from one execution or load region might use a string stored in different region. If you do not want this behavior, then do one of the following:

- use the `PROTECTED` load region attribute if you are using scatter-loading
- globally disable merging with `--no_merge`.

2.76.2 See also

Reference

- *--info=topic[,topic,...]* on page 2-59
- *Load region attributes* on page 4-7.

2.77 `--muldefweak`, `--no_muldefweak`

This option enables or disables multiple weak definitions of a symbol.

If enabled, the linker chooses the first definition that it encounters and discards all the other duplicate definitions. If disabled, the linker generates an error message for all multiply defined weak symbols.

2.77.1 Default

The default is `--no_muldefweak`.

2.78 `--output=file`

This option specifies the name of the output file. The file can be either a partially-linked object or an executable image, depending on the command-line options used.

2.78.1 Syntax

`--output=file`

If `--output=file` is not specified, the linker uses the following default filenames:

`__image.axf`	if the output is an executable image
`__object.o`	if the output is a partially-linked object.

If *file* is specified without path information, it is created in the current working directory. If path information is specified, then that directory becomes the default output directory.

2.78.2 See also

Reference

- *--callgraph_file=filename* on page 2-17
- *--partial* on page 2-94.

2.79 `--override_visibility`

This option enables `EXPORT` and `IMPORT` directives in a steering file to override the visibility of a symbol.

By default:

- only symbol definitions with `STV_DEFAULT` or `STV_PROTECTED` visibility can be exported
- only symbol references with `STV_DEFAULT` visibility can be imported.

When you specify `--override_visibility`, any global symbol definition can be exported and any global symbol reference can be imported.

2.79.1 See also

Reference

- *--keep_protected_symbols* on page 2-70
- *--undefined_and_export=symbol* on page 2-135
- *EXPORT* on page 3-2
- *IMPORT* on page 3-4.

2.80 `--pad=`*`num`*

This option enables you to set a value for padding bytes. The linker assigns this value to all padding bytes inserted in load or execution regions.

2.80.1 Syntax

`--pad=`*`num`*

Where *`num`* is an integer, which can be given in hexadecimal format. For example, setting *`num`* to `0xFF` might help to speed up ROM programming time. If *`num`* is greater than `0xFF`, then the padding byte is cast to a `char`, that is `(char)`*`num`*.

——— **Note** ———

Padding is only inserted:

- Within load regions. No padding is present *between* load regions.
- Between fixed execution regions (in addition to forcing alignment). Padding is not inserted up to the maximum length of a load region unless it has a fixed execution region at the top.
- Between sections to ensure that they conform to alignment constraints.

———————————

2.80.2 See also

Concepts

- *Input sections, output sections, regions, and Program Segments* on page 4-5
- *Load view and execution view of an image* on page 4-6.

2.81 `--paged`

This option enables Demand Paging mode to help produce ELF files that can be demand paged efficiently.

A default page size of `0x8000` bytes is used. You can change this with the `--pagesize` command-line option.

2.81.1 See also

Concepts

Using the Linker:

- *Demand paging* on page 4-23
- *About creating regions on page boundaries* on page 8-52.

Reference

- *--pagesize=pagesize* on page 2-93

2.82 `--pagesize=`*`pagesize`*

This option enables you to change the page size used when demand paging.

2.82.1 Syntax

```
--pagesize=pagesize
```

Where *`pagesize`* is the page size in bytes. The default value is `0x8000`.

2.82.2 See also

Concepts

Using the Linker:

- *Demand paging* on page 4-23
- *About creating regions on page boundaries* on page 8-52.

Reference

- *--paged* on page 2-92.

2.83 `--partial`

This option creates a partially-linked object that can be used in a subsequent link step.

2.83.1 See also

Concepts

Using the Linker:

- *Partial linking model* on page 3-4.

2.84 `--piveneer`, `--no_piveneer`

This option enables or disables the generation of a veneer for a call from *position independent* (PI) code to absolute code. When using `--no_piveneer`, an error message is produced if the linker detects a call from PI code to absolute code.

2.84.1 Default

The default is `--piveneer`.

2.84.2 See also

Concepts

Using the Linker:

- *Overview of veneers* on page 4-26
- *Veneer sharing* on page 4-27
- *Veneer types* on page 4-28
- *Generation of position independent to absolute veneers* on page 4-29
- *Reuse of veneers when scatter-loading* on page 4-30.

Reference

- *--inlineveneer, --no_inlineveneer* on page 2-65
- *--veneershare, --no_veneershare* on page 2-141.

2.85 --predefine="*string*"

When preprocessing the scatter file, this option enables commands to be passed to the pre-processor. You specify a pre-processor on the first line of the scatter file.

2.85.1 Syntax

`--predefine="string"`

You can use more than one `--predefine` option on the command-line.

You can also use the synonym:`--pd="string"`.

2.85.2 Restrictions

Use this option with `--scatter`.

2.85.3 Example

The following example shows the scatter file contents before pre-processing.

Example 2-1 Scatter file before pre-processing

```
#! armcc -E
lr1 BASE
{
    er1 BASE
    {
        *(+RO)
    }
    er2 BASE2
    {
        *(+RW+ZI)
    }
}
```

Use `armlink` with the command-line options:

```
--predefine="-DBASE=0x8000" --predefine="-DBASE2=0x1000000" --scatter=file
```

This passes the command-line options: `-DBASE=0x8000 -DBASE2=0x1000000` to the compiler to pre-process the scatter file.

The following example shows how the scatter file looks after pre-processing:

Example 2-2 Scatter file after pre-processing

```
lr1 0x8000
{
    er1 0x8000
    {
        *(+RO)
    }
    er2 0x1000000
    {
        *(+RW+ZI)
    }
}
```

2.85.4 See also

Concepts

Using the Linker:

- *Using preprocessing commands in a scatter file* on page 8-55.

Reference

- *--scatter=file* on page 2-110.

2.86 `--privacy`

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

When producing an executable image it removes local symbols from the output symbol table.

For object files built with the `--partial` option, 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.

2.86.1 See also

Reference

- *--locals, --no_locals* on page 2-80
- *--partial* on page 2-94.

2.87 `--reduce_paths`, `--no_reduce_paths`

This option enables or disables the elimination of redundant path name information in file paths.

2.87.1 Mode

Effective on Windows systems only.

2.87.2 Default

The default is `--no_reduce_paths`.

2.87.3 Usage

Windows systems impose a 260 character limit on file paths. Where path names exist whose absolute names expand to longer than 260 characters, you can use the `--reduce_paths` option to reduce absolute path name length by matching up directories with corresponding instances of `..` and eliminating the directory/`..` sequences in pairs.

Note

It is recommended that you avoid using long and deeply nested file paths, in preference to minimizing path lengths using the `--reduce_paths` option.

2.87.4 Example

A file to be linked might be at the location:

```
..\..\..\xyzzy\xyzzy\objects\file.c
```

Your current working directory might be at the location:

```
\foo\bar\baz\gazonk\quux\bop
```

The combination of these paths results in the path:

```
\foo\bar\baz\gazonk\quux\bop\..\..\..\xyzzy\xyzzy\objects\file.o
```

By using the option `--reduce_paths` the path becomes:

```
\foo\bar\baz\xyzzy\xyzzy\objects\file.c
```

2.88 `--ref_cpp_init`, `--no_ref_cpp_init`

This option enables or disables the linker adding a reference to the C++ static object initialization routine in the ARM libraries. The default reference added is `__cpp_initialize__aeabi_`. To change this you can use `--cppinit`.

2.88.1 Default

The default is `--ref_cpp_init`.

2.88.2 See also

Concepts

Using C and C++ Libraries and Floating-Point Support:

- *C++ initialization, construction and destruction* on page 2-56.

Reference

- *--cppinit, --no_cppinit* on page 2-25.

2.89 `--reloc`

This option creates a single relocatable load region with contiguous execution regions.

2.89.1 Usage

Only use this option for legacy systems with the type of relocatable ELF images that conform to the *ELF for the ARM Architecture* specification. The generated image might not be compliant with the ELF for the ARM Architecture specification.

When relocated `MOVT` and `MOVW` instructions are encountered in an image being linked with `--reloc`, `armlink` produces the following additional dynamic tags:

DT_RELA	The address of a relocation table.
DT_RELASZ	The total size, in bytes, of the DT_RELA relocation table.
DT_RELAENT	The size, in bytes, of the DT_RELA relocation entry.

2.89.2 See also

Concepts

Using the Linker:

- *Type 1 image, one load region and contiguous execution regions* on page 8-59
- *Type 3 image, two load regions and non-contiguous execution regions* on page 8-63.

Other information

- *Base Platform ABI for the ARM Architecture*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0037-/index.html
- *ELF for the ARM Architecture*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044-/index.html.

2.90 `--remarks`

This option forces the linker to display remarks that are otherwise hidden by default when used with the `--diag_remarks` option.

——— **Note** ———

The linker does not issue remarks by default.

———————————

2.90.1 See also

Reference

- *--diag_remark=tag[,tag,...]* on page 2-32
- *--errors=file* on page 2-44.

2.91 `--remove, --no_remove`

This option enables or disables the removal of unused input sections from the image. An input section is considered used if it contains an entry point, or if it is referred to from a used section.

2.91.1 Default

The default is `--remove`.

2.91.2 Usage

By default, unused section elimination is disabled when building *dynamically linked libraries* (DLLs) or shared objects, Use `--remove` to re-enable unused section elimination.

Use `--no_remove` when debugging to retain all input sections in the final image even if they are unused.

Use `--remove` with the `--keep` option to retain specific sections in a normal build.

2.91.3 See also

Concepts

Using the Linker:

- *Elimination of common debug sections* on page 5-2
- *Elimination of common groups or sections* on page 5-3
- *Elimination of unused sections* on page 5-4
- *Elimination of unused virtual functions* on page 5-5.

Reference

- *--keep=section_id* on page 2-68

2.92 `--ro_base=`*`address`*

This option sets both the load and execution addresses of the region containing the RO output section at a specified address.

2.92.1 Syntax

`--ro_base=`*`address`*

Where *`address`* must be word-aligned.

2.92.2 Default

If this option is not specified, and no scatter file is specified, the default is `--ro_base=0x8000`.

2.92.3 Restrictions

You cannot use `--ro_base` with `--scatter`.

2.92.4 See also

Reference

- *--ropi* on page 2-105
- *--rosplit* on page 2-106
- *--rw_base=address* on page 2-107
- *--rwpi* on page 2-108
- *--scatter=file* on page 2-110
- *--zi_base=address* on page 2-150.

2.93 `--ropi`

This option makes the load and execution region containing the RO output section position-independent. If this option is not used, the region is marked as absolute. Usually each read-only input section must be *Read-Only Position-Independent* (ROPI). If this option is selected, the linker:

- checks that relocations between sections are valid
- ensures that any code generated by the linker itself, such as interworking veneers, is ROPI.

Note

The linker gives a downgradable error if `--ropi` is used without `--rwpi` or `--rw_base`.

2.93.1 Restrictions

You cannot use `--ropi` with `--scatter`.

2.93.2 See also

Reference

- *--ro_base=address* on page 2-104
- *--rosplit* on page 2-106
- *--rw_base=address* on page 2-107
- *--rwpi* on page 2-108
- *--scatter=file* on page 2-110

2.94 `--rosplit`

This option splits the default RO load region into two RO output sections, one for `RO-CODE` and one for `RO-DATA`.

2.94.1 Restrictions

You cannot use `--rosplit` with `--scatter`.

2.94.2 See also

Reference

- *--ro_base=address* on page 2-104
- *--ropi* on page 2-105
- *--rw_base=address* on page 2-107
- *--rwpi* on page 2-108
- *--scatter=file* on page 2-110

2.95 `--rw_base=`*`address`*

This option sets the execution addresses of the region containing the RW output section at a specified address.

2.95.1 Syntax

`--rw_base=`*`address`*

Where *`address`* must be word-aligned.

2.95.2 Restrictions

You cannot use `--rw_base` with `--scatter`.

2.95.3 See also

Reference

- *--ro_base=address* on page 2-104
- *--ropi* on page 2-105
- *--rosplit* on page 2-106
- *--rwpi* on page 2-108
- *--scatter=file* on page 2-110
- *--split* on page 2-118
- *--zi_base=address* on page 2-150.

2.96 `--rwpi`

This option makes the load and execution region containing the RW and ZI output section position-independent. If this option is not used the region is marked as absolute. This option requires a value for `--rw_base`. If `--rw_base` is not specified, `--rw_base=0` is assumed. Usually each writable input section must be *read-write position-independent* (RWPI).

If this option is selected, the linker:

- checks that the `PI` attribute is set on input sections to any read-write execution regions
- checks that relocations between sections are valid
- generates entries relative to the static base in the table `Region$$Table`.

  This is used when regions are copied, decompressed, or initialized.

2.96.1 Restrictions

You cannot use `--rwpi` with `--scatter`.

2.96.2 See also

Reference

- *--ro_base=address* on page 2-104
- *--ropi* on page 2-105
- *--rosplit* on page 2-106
- *--rw_base=address* on page 2-107
- *--scatter=file* on page 2-110
- *--split* on page 2-118

2.97 `--scanlib`, `--no_scanlib`

This option enables or disables scanning of the ARM libraries to resolve references. Use `--no_scanlib` if you want to link your own libraries.

2.97.1 Default

The default is `--scanlib`.

2.98 `--scatter=file`

This option creates an image memory map using the scatter-loading description contained in the specified file. The description provides grouping and placement details of the various regions and sections in the image.

2.98.1 Syntax

`--scatter=file`

Where `file` is the name of a scatter file.

2.98.2 Usage

To modify the placement of any unassigned input sections when `.ANY` selectors are present, use the following command-line options with `--scatter`:

- `--any_contingency`
- `--any_placement`
- `--any_sort_order`
- `--tiebreaker`.

The `--scatter` option cannot be used with `--first`, `--last`, `--partial`, `--reloc`, `--ro_base`, `--ropi`, `--rosplit`, `--rw_base`, `--rwpi`, `--split`, `--startup`, and `--zi_base`.

2.98.3 See also

Concepts

- *Behavior when .ANY sections overflow because of linker-generated content* on page 4-28.

Using the Linker:

- Chapter 8 *Using scatter files*.

Reference

- *--any_contingency* on page 2-5
- *--any_placement=algorithm* on page 2-6
- *--any_sort_order=order* on page 2-8
- *--first=section_id* on page 2-53
- *--last=section_id* on page 2-72
- *--partial* on page 2-94
- *--reloc* on page 2-101
- *--ro_base=address* on page 2-104
- *--ropi* on page 2-105
- *--rosplit* on page 2-106
- *--rw_base=address* on page 2-107
- *--rwpi* on page 2-108
- *--split* on page 2-118
- *--startup=symbol, --no_startup* on page 2-119
- *--tiebreaker=option* on page 2-133
- *--zi_base=address* on page 2-150.

2.99 `--section_index_display=type`

This option changes the display of the index column when printing memory map output. Use this option with `--map`.

2.99.1 Syntax

`--section_index_display=type`

Where *type* is one of the following:

`cmdline`	Alters the display of the map file to show the order that a section appears on the command-line. The command-line order is defined as `File.Object.Section` where: • `Section` is the section index, `sh_idx`, of the `Section` in the `Object` • `Object` is the order that `Object` appears in the `File` • `File` is the order the `File` appears on the command line. The order the `Object` appears in the `File` is only significant if the file is an `ar` archive.
`internal`	The index value represents the order in which the linker creates the section.
`input`	The index value represents the section index of the section in the original input file.

2.99.2 Usage

Use `--map` with `--section_index_display=input` when you want to find the exact section in an input object.

2.99.3 Default

The default is `--section_index_display=internal`.

2.99.4 See also

Reference

- *--map, --no_map* on page 2-83
- *--tiebreaker=option* on page 2-133.

2.100 `--show_cmdline`

This option outputs the command-line used by the linker. It shows the command-line after processing by the linker, and can be useful to check:

- the command-line a build system is using
- how the linker 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 output stream (`stdout`).

2.100.1 See also

Reference

- *--help* on page 2-58
- *--via=file* on page 2-145.

2.101 `--show_full_path`

If the file representing object `obj` has full path name `path/to/obj` then the linker displays `path/to/obj` instead of `obj` in any diagnostic.

2.101.1 See also

Reference

- *--show_parent_lib* on page 2-114
- *--show_sec_idx* on page 2-115.

2.102 `--show_parent_lib`

If an object `obj` comes from library `lib`, then displays `lib(obj)` instead of `obj` in any diagnostic.

2.102.1 See also

Reference

- *--show_full_path* on page 2-113
- *--show_sec_idx* on page 2-115.

2.103 `--show_sec_idx`

Displays the section index, `sh_idx`, of section in the originating object.

For example, if section `sec` has section index 3 then it is displayed as `sec:3` in all diagnostics

2.103.1 See also

Reference

- *--show_full_path* on page 2-113
- *--show_parent_lib* on page 2-114.

2.104 `--sort=`*`algorithm`*

This option specifies the sorting algorithm used to determine the order of sections in an output image. The sorting algorithms conform to the standard rules placing input section in ascending order by attributes.

Sort algorithms can also be specified in a scatter file for individual execution regions using the `SORTTYPE` keyword.

2.104.1 Syntax

`--sort=`*`algorithm`*

Where *`algorithm`* is one of the following:

`Alignment` Sorts input sections by ascending order of alignment value.

`AvgCallDepth` Sorts all Thumb code before ARM code and then sorts according to the approximated average call depth of each section in ascending order.

Use this algorithm to minimize the number of long branch veneers.

Note

The approximation of the average call depth depends on the order of input sections. Therefore, this sorting algorithm is more dependent on the order of input sections than using, say, `RunningDepth`.

`BreadthFirstCallTree`

This is similar to the `CallTree` algorithm except that it uses a breadth-first traversal when flattening the Call Tree into a list.

`CallTree` The linker flattens the call tree into a list containing the read-only code sections from all execution regions that have `CallTree` sorting enabled.

Sections in this list are copied back into their execution regions, followed by all the non read-only code sections, sorted lexically. Doing this ensures that sections calling each other are placed close together.

Note

This sorting algorithm is less dependent on the order of input sections than using either `RunningDepth` or `AvgCallDepth`.

`Lexical` Sorts according to the name of the section and then by input order if the names are the same.

`LexicalState` Sorts Thumb code before ARM code, then sorts lexically.

`List` Provides a list of the available sorting algorithms. The linker terminates after displaying the list.

`ObjectCode` Sorts code sections by tiebreaker. All other sections are sorted lexically. This is most useful when used with `--tiebreaker=cmdline` because it attempts to group all the sections from the same object together in the memory map.

`RunningDepth` Sorts all Thumb code before ARM code and then sorts according to the running depth of the section in ascending order. The running depth of a section S is the average call depth of all the sections that call S, weighted by the number of times that they call S.

Use this algorithm to minimize the number of long branch veneers.

2.104.2 Default

The default algorithm is `--sort=Lexical`. In large-region mode, the default algorithm is `--sort=AvgCallDepth`.

2.104.3 See also

Concepts

- *About execution region descriptions* on page 4-8.

Using the Linker:

- *Section placement with the linker* on page 4-19.

Reference

- *--largeregions, --no_largeregions* on page 2-71
- *Execution region attributes* on page 4-11.

2.105 `--split`

This option splits the default load region, that contains the RO and RW output sections, into the following load regions:

- One region containing the RO output section. The default load address is `0x8000`, but a different address can be specified with the `--ro_base` option.
- One region containing the RW and ZI output sections. The load address is specified with the `--rw_base` option. This option requires a value for `--rw_base`. If `--rw_base` is not specified, `--rw_base=0` is assumed.

Both regions are root regions.

2.105.1 Restrictions

You cannot use `--split` with `--scatter`.

2.105.2 See also

Concepts

Using the Linker:

- *The image structure* on page 4-3.

Reference

- *--ro_base=address* on page 2-104
- *--rw_base=address* on page 2-107
- *--scatter=file* on page 2-110

2.106 `--startup=`*`symbol`*, `--no_startup`

This option enables the linker to use alternative C libraries with a different startup symbol if required.

2.106.1 Syntax

`--startup=`*`symbol`*

By default, *`symbol`* is set to `__main`.

`--no_startup` does not take a *symbol* argument.

2.106.2 Default

The default is `--startup=__main`.

2.106.3 Usage

The linker includes the C library startup code if there is a reference to a symbol that is defined by the C library startup code. This symbol reference is called the startup symbol. It is automatically created by the linker when it sees a definition of main(). The `--startup` option enables you to change this symbol reference.

- If the linker finds a definition of `main()` and does not find a reference to (or definition of) *`symbol`*, then it generates an error.
- If the linker finds a definition of `main()` and a reference to (or definition of) *`symbol`*, and no entry point is specified, then the linker generates a warning.

2.106.4 See also

Reference

- *--entry=location* on page 2-42.

2.107 `--strict`

This option instructs the linker to perform additional conformance checks, such as reporting conditions that might result in failures. An example of such a condition is taking the address of an interworking function from a non-interworking function.

2.107.1 Usage

`--strict` causes the linker to check for taking the address of:

- A non-interworking location from a non-interworking location in a different state.
- A RWPI location from a location that uses the static base register R9.
- A stack checked location from a location that uses the reserved stack checking register R10. (This is for ADS compatibility only.)
- A location that uses the reserved stack checking register r10 from a stack checked location. (This is for ADS compatibility only).

2.107.2 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--diag_error=tag[,tag,...]* on page 2-31
- *--diag_suppress=tag[,tag,...]* on page 2-34
- *--diag_warning=tag[,tag,...]* on page 2-35
- *--errors=file* on page 2-44
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

Assembler Reference:

- *--diag_error=tag{, tag}* on page 2-9
- *--diag_suppress=tag{, tag}* on page 2-10
- *--diag_warning=tag{, tag}* on page 2-11.

Compiler Reference:

- *--diag_error=tag[,tag,...]* on page 3-30
- *--diag_suppress=tag[,tag,...]* on page 3-32
- *--diag_warning=tag[,tag,...]* on page 3-33
- *--strict, --no_strict* on page 3-88
- *--strict_warnings* on page 3-89.

2.108 `--strict_enum_size`, `--no_strict_enum_size`

The option `--strict_enum_size` causes the linker to display an error message if the enum size is not consistent across all inputs. This is the default.

Use `--no_strict_enum_size` for compatibility with objects built using RVCT v3.1 and earlier.

2.108.1 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

Compiler Reference:

- *--enum_is_int* on page 3-36.

2.109 `--strict_flags`, `--no_strict_flags`

The option `--strict_flags` prevents the EF_ARM_HASENTRY flag from being generated.

2.109.1 Default

The default is `--no_strict_flags`.

2.109.2 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

Other information

- *ARM ELF Specification (SWS ESPC 0003 B-02)*, http://infocenter.arm.com/help/topic/com.arm.doc.espc0003/index.html.

2.110 `--strict_ph`, `--no_strict_ph`

The linker writes the contents of load regions into the output ELF file in the order that load regions are written in the scatter file. Each load region is represented by one ELF program segment. In RVCT v2.2 the Program Header Table entries describing the program segments are given the same order as the program segments in the ELF file. To be more compliant with the ELF specification, in RVCT v3.0 and later the Program Header Table entries are sorted in ascending virtual address order.

Use the `--no_strict_ph` command-line option to switch off the sorting of the Program Header Table entries.

2.110.1 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

2.111 `--strict_relocations`, `--no_strict_relocations`

This option enables you to ensure *Application Binary Interface* (ABI) compliance of legacy or third party objects. It checks that branch relocation applies to a branch instruction bit-pattern. The linker generates an error if there is a mismatch.

2.111.1 Usage

Use `--strict_relocations` to instruct the linker to report instances of obsolete and deprecated relocations.

Relocation errors and warnings are most likely to occur if you are linking object files built with previous versions of the ARM tools.

2.111.2 Default

The default is `--no_strict_relocations`.

2.111.3 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

2.112 --strict_symbols, --no_strict_symbols

The option `--strict_symbols` checks that the mapping symbol type matches ABI symbol type. The linker displays a warning if the types do not match.

A mismatch can occur only if you have hand-coded your own assembler.

2.112.1 Default

The default is `--no_strict_symbols`.

2.112.2 Example

In the following assembler code the symbol `sym` has type `STT_FUNC` and is ARM:

```
    area code, readonly
        DCD sym + 4
        ARM
sym  PROC
        NOP
        THUMB
        NOP
        ENDP
        END
```

The difference in behavior is the meaning of `DCD sym + 4`:

- In pre-ABI linkers the state of the symbol is the state of the only of the mapping symbol at that location. In this example, the state is Thumb.
- In ABI linkers the type of the symbol is the state of the location of symbol plus the offset.

2.112.3 See also

Concepts

Using the Linker:
- *Use of the strict family of options in the linker* on page 4-40
- *About mapping symbols* on page 7-3.

Reference
- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_visibility, --no_strict_visibility* on page 2-126
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

2.113 `--strict_visibility`, `--no_strict_visibility`

A linker is not permitted to match a symbol reference with `STT_HIDDEN` visibility to a dynamic shared object. Some older linkers might permit this.

Use `--no_strict_visibility` to permit a hidden visibility reference to match against a shared object.

2.113.1 Default

The default is `--strict_visibility`.

2.113.2 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_wchar_size, --no_strict_wchar_size* on page 2-127.

2.114 `--strict_wchar_size`, `--no_strict_wchar_size`

The option `--strict_wchar_size` causes the linker to display an error message if the wide character size is not consistent across all inputs. This is the default.

Use `--no_strict_wchar_size` for compatibility with objects built using RVCT v3.1 and earlier.

2.114.1 See also

Concepts

Using the Linker:

- *Use of the strict family of options in the linker* on page 4-40.

Reference

- *--strict* on page 2-120
- *--strict_enum_size, --no_strict_enum_size* on page 2-121
- *--strict_flags, --no_strict_flags* on page 2-122
- *--strict_ph, --no_strict_ph* on page 2-123
- *--strict_relocations, --no_strict_relocations* on page 2-124
- *--strict_symbols, --no_strict_symbols* on page 2-125
- *--strict_visibility, --no_strict_visibility* on page 2-126.

Compiler Reference:

- *--wchar16* on page 3-97
- *--wchar32* on page 3-97.

2.115 `--symbolic`

Sets the DF_SYMBOLIC flag in the SHT_DYNAMIC section for a shared library. This flag changes the symbol resolution algorithm of the dynamic linker for references within the library. The dynamic linker searches for symbols starting with the shared object rather than the executable image. If the referenced symbol cannot be found in the shared object, the dynamic linker searches the executable image and other shared objects as usual.

2.116 `--symbols`, `--no_symbols`

This option enables or disables the listing of each local and global symbol used in the link step, and its value.

Note

This does not include mapping symbols output to `stdout`. Use `--list_mapping_symbols` to include mapping symbols in the output.

2.116.1 Default

The default is `--no_symbols`.

2.116.2 See also

Reference

- *--list_mapping_symbols, --no_list_mapping_symbols* on page 2-78.

2.117 `--symdefs=file`

This option creates a file containing the global symbol definitions from the output image.

2.117.1 Syntax

`--symdefs=file`

where *`file`* is the name of the text file to contain the global symbol definitions.

2.117.2 Default

By default, all global symbols are written to the symdefs file. If a symdefs file called *`file`* already exists, the linker restricts its output to the symbols already listed in this file.

——— **Note** ———

If you do not want this behavior, be sure to delete any existing symdefs file before the link step.

———————————

2.117.3 Usage

If *`file`* is specified without path information, the linker searches for it in the directory where the output image is being written. If it is not found, it is created in that directory.

You can use the symbol definitions file as input when linking another image.

2.117.4 See also

Concepts

Using the Linker:

- *Accessing symbols in another image* on page 7-17.

2.118 `--tailreorder`, `--no_tailreorder`

This option moves tail calling sections immediately before their target, if possible, to optimize the branch instruction at the end of a section. A tail calling section is a section that contains a branch instruction at the end of the section. The branch must have a relocation that targets a function at the start of a section.

2.118.1 Default

The default is `--no_tailreorder`.

2.118.2 Restrictions

The linker:

- Can only move one tail calling section for each tail call target. If there are multiple tail calls to a single section, the tail calling section with an identical section name is moved before the target. If no section name is found in the tail calling section that has a matching name, then the linker moves the *first* section it encounters.
- Cannot move a tail calling section out of its execution region.
- Does not move tail calling sections before inline veneers.

2.118.3 See also

Concepts

Using the Linker:

- *Handling branches that optimize to a NOP* on page 5-20
- *About reordering of tail calling sections* on page 5-21.

Reference

- *--branchnop, --no_branchnop* on page 2-14.

2.119 `--thumb2_library`, `--no_thumb2_library`

Enables you to link against the combined ARM and Thumb-2 library for use with Cortex-R series processors.

Use the `--no_thumb2_library` option to revert to the ARMv5T and later libraries.

2.119.1 Default

The default is `--thumb2_library`.

2.119.2 See also

Reference

Using ARM C and C++ Libraries and Floating-Point Support:

- *C and C++ library naming conventions* on page 2-120.

2.120 `--tiebreaker=option`

A tiebreaker is used when a sorting algorithm requires a total ordering of sections. It is used to resolve the order when the sorting criteria results in more than one input section with equal properties.

2.120.1 Syntax

`--tiebreaker=option`

where *`option`* is one of:

`creation` The order that the linker creates sections in its internal section data structure.

When the linker creates an input section for each ELF section in the input objects, it increments a global counter. The value of this counter is stored in the section as the creation index.

The creation index of a section is unique apart from the special case of inline veneers.

`cmdline` The order that the section appears on the linker command-line. The command-line order is defined as `File.Object.Section` where:

- `Section` is the section index, `sh_idx`, of the `Section` in the `Object`
- `Object` is the order that `Object` appears in the `File`
- `File` is the order the `File` appears on the command line.

The order the `Object` appears in the `File` is only significant if the file is an `ar` archive.

This option is useful if you are doing a binary difference between the results of different links, link1 and link2. If link2 has only small changes from link1, then you might want the differences in one source file to be localized. In general, creation index works well for objects, but because of the multiple pass selection of members from libraries, a small difference such as calling a new function can result in a different order of objects and therefore a different tiebreak. The command-line index is more stable across builds.

Use this option with the `--scatter` option.

2.120.2 Default

The default option is `creation`.

2.120.3 See also

Concepts

Using the Linker:

- *Examples of using sorting algorithms for .ANY sections* on page 8-30.

Reference

- *--any_sort_order=order* on page 2-8
- *--map, --no_map* on page 2-83
- *--scatter=file* on page 2-110
- *--section_index_display=type* on page 2-111
- *--sort=algorithm* on page 2-116.

2.121 `--undefined=`*`symbol`*

This option causes the linker to:

1. Create a symbol reference to the specified symbol name.
2. Issue an implicit `--keep(`*`symbol`*`)` to prevent any sections brought in to define that symbol from being removed.

2.121.1 Syntax

`--undefined=`*`symbol`*

2.121.2 See also

Reference

- *--keep=section_id* on page 2-68
- *--undefined_and_export=symbol* on page 2-135.

2.122 `--undefined_and_export=`*`symbol`*

This option causes the linker to:

1. Create a symbol reference to the specified symbol name.
2. Issue an implicit `--keep(`*`symbol`*`)` to prevent any sections brought in to define that symbol from being removed.
3. Add an implicit `EXPORT` *`symbol`* to push the specified symbol into the dynamic symbol table.

2.122.1 Syntax

```
--undefined_and_export=symbol
```

2.122.2 Usage

Be aware of the following when using this option:

- It does not change the visibility of a symbol unless you specify the `--override_visibility` option.
- A warning is issued if the visibility of the specified symbol is not high enough.
- A warning is issued if the visibility of the specified symbol is overridden because you also specified the `--override_visibility` option.
- Hidden symbols are not exported unless you specify the `--override_visibility` option.

2.122.3 See also

Reference

- *--keep=section_id* on page 2-68
- *--override_visibility* on page 2-90
- *--undefined=symbol* on page 2-134
- *EXPORT* on page 3-2.

2.123 `--unresolved=`*`symbol`*

This option takes each reference to an undefined symbol and matches it to the global definition of the specified symbol.

2.123.1 Syntax

`--unresolved=`*`symbol`*

Where *`symbol`* must be both defined and global, otherwise it appears in the list of undefined symbols and the link step fails.

2.123.2 Usage

This option is particularly useful during top-down development, because it enables you to test a partially-implemented system by matching each reference to a missing function to a dummy function.

2.123.3 See also

Reference

- *--undefined=symbol* on page 2-134
- *--undefined_and_export=symbol* on page 2-135

2.124 `--use_definition_visibility`

When the linker combines global symbols the visibility of the symbol is set with the strictest visibility of the symbols being combined. Therefore, a symbol reference with `STV_HIDDEN` visibility combined with a definition with `STV_DEFAULT` visibility results in a definition with `STV_HIDDEN` visibility.

This option enables the linker to use the visibility of the definition in preference to the visibility a reference when combining symbols. For example, a symbol reference with `STV_HIDDEN` visibility combined with a definition with `STV_DEFAULT` visibility results in a definition with `STV_DEFAULT` visibility.

This can be useful when you want a reference to not match a Shared Library, but you want to export the definition.

Note

This option is not ELF-compliant and is disabled by default. To create ELF-compliant images, you must use symbol references with the appropriate visibility.

2.125 `--userlibpath=`*`pathlist`*

This option specifies a list of paths that are used to search for user libraries.

2.125.1 Syntax

`--userlibpath=`*`pathlist`*

Where *`pathlist`* is a comma-separated list of paths that are used to search for the required libraries. Do not include spaces between the comma and the path name when specifying multiple path names, for example, *`path1,path2,path3,...,pathn`*.

2.125.2 See also

Concepts

Using the Linker:

- *How the linker performs library searching, selection, and scanning* on page 4-35.

Reference

- *--libpath=pathlist* on page 2-75.

2.126 `--veneer_inject_type=`*`type`*

This option controls the veneer layout when `--largeregions` mode is on.

2.126.1 Syntax

`--veneer_inject_type=`*`type`*

where *`type`* is one of:

`individual` The linker places veneers to ensure they can be reached by the largest amount of sections that use the veneer. Veneer reuse between execution regions is permitted. This type minimizes the number of veneers that are required but disrupts the structure of the image the most.

`pool` The linker:

1. Collects veneers from a contiguous range of the execution region
2. Places all the veneers generated from that range into a pool.
3. Places that pool at the end of the range.

A large execution region might have more than one range and therefore more than one pool. Although this type has much less impact on the structure of image, it has fewer opportunities for reuse. This is because a range of code cannot reuse a veneer in another pool. The linker calculates the range based on the presence of branch instructions that the linker predicts might require veneers. A branch is predicted to require a veneer when either:

- a state change is required
- the distance from source to target plus a contingency greater than the branch range.

You can set the size of the contingency with the `--veneer_pool_size=`*`size`* option. By default the contingency size is set to 102400 bytes. The `--info=veneerpools` option provides information on how the linker has placed veneer pools.

2.126.2 Restrictions

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

2.126.3 See also

Reference

- *--info=topic[,topic,...]* on page 2-59
- *--largeregions, --no_largeregions* on page 2-71
- *--veneer_pool_size=size* on page 2-140.

2.127 `--veneer_pool_size=`*`size`*

Sets the contingency size for the veneer pool in an execution region.

2.127.1 Syntax

```
--veneer_pool_size=pool
```

where *`pool`* is the size in bytes.

2.127.2 Default

The default size is 102400 bytes.

2.127.3 See also

Reference

- *--veneer_inject_type=type* on page 2-139.

2.128 `--veneershare, --no_veneershare`

This option enables or disables veneer sharing. Veneer sharing can cause a significant decrease in image size.

2.128.1 default

The default is `--veneershare`.

2.128.2 See also

Concepts

Using the Linker:

- *Overview of veneers* on page 4-26
- *Veneer sharing* on page 4-27
- *Veneer types* on page 4-28
- *Generation of position independent to absolute veneers* on page 4-29
- *Reuse of veneers when scatter-loading* on page 4-30.

Reference

- *--inlineveneer, --no_inlineveneer* on page 2-65
- *--piveneer, --no_piveneer* on page 2-95.

2.129 `--verbose`

This option prints detailed information about the link operation, including the objects that are included and the libraries from which they are taken. This output is particular useful for tracing undefined symbols reference or multiply defined symbols. Because this output is typically quite long, you might want to use this command with the `--list=`*`file`* command to redirect the information to *`file`*.

Use `--verbose` to output diagnostics to `stdout`.

2.129.1 See also

Reference

- *--list=file* on page 2-77
- *--muldefweak, --no_muldefweak* on page 2-88
- *--unresolved=symbol* on page 2-136.

2.130 `--version_number`

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

2.130.1 Syntax

```
armlink --version_number
```

The linker displays the version number in the format `nnnbbb`, where:

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

2.130.2 Example

Version 4.1.0 build 713 is displayed as `410713`.

2.130.3 See also

Reference

- *--help* on page 2-58
- *--vsn* on page 2-146

2.131 `--vfemode=mode`

Virtual Function Elimination (VFE) is a technique that enables the linker to identify more unused sections.

Use this option to specify how VFE, and *Runtime Type Information* (RTTI) objects, are eliminated.

2.131.1 Syntax

`--vfemode=mode`

Where *`mode`* is one of the following:

`on`
: Use the command-line option `--vfemode=on` to make the linker VFE aware.

  In this mode the linker chooses `force` or `off` mode based on the content of object files:

  - Where every object file contains VFE information or does not refer to a symbol with a mangled C++ name, the linker assumes `force` mode and continues with the elimination.
  - If any object file is missing VFE information and refers to a symbol with a mangled C++ name, for example, where code has been compiled with a previous release of the ARM tools, the linker assumes `off` mode, and VFE is disabled silently. Choosing `off` mode to disable VFE in this situation ensures that the linker does not remove a virtual function that is used by an object with no VFE information.

`off`
: Use the command-line option `--vfemode=off` to make `armlink` ignore any extra information supplied by the compiler. In this mode, the final image is the same as that produced by compiling and linking without VFE awareness.

`force`
: Use the command-line option `--vfemode=force` to make the linker VFE aware and force the VFE algorithm to be applied. If some of the object files do not contain VFE information, for example, where they have been compiled with a previous release of the ARM tools, the linker continues with the elimination but displays a warning to alert you to possible errors.

`force_no_rtti`
: Use the command-line option `--vfemode=force_no_rtti` to make the linker VFE aware and force the removal of all RTTI objects. In this mode all virtual functions are retained.

2.131.2 Default

The default is `--vfemode=on`.

2.131.3 See also

Concepts

Using the Linker:

- *Elimination of common debug sections* on page 5-2
- *Elimination of common groups or sections* on page 5-3
- *Elimination of unused sections* on page 5-4
- *Elimination of unused virtual functions* on page 5-5.

2.132 `--via=`*`file`*

This option reads an additional list of input filenames and linker options from *`file`*.

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

2.132.1 See also

Concepts

Compiler Reference:

- *Overview of via files* on page B-2.

2.133 --vsn

This option displays the version information and the license details. For example:

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

2.133.1 See also

Reference

- *--help* on page 2-58
- *--show_cmdline* on page 2-112
- *--version_number* on page 2-143.

2.134 `--xref`, `--no_xref`

This option lists to stdout all cross-references between input sections.

2.134.1 Default

The default is `--no_xref`.

2.134.2 See also

Reference

- *--list=file* on page 2-77
- *--xrefdbg, --no_xrefdbg* on page 2-148
- *--xref{from|to}=object(section)* on page 2-149.

2.135 `--xrefdbg`, `--no_xrefdbg`

This option lists to stdout all cross-references between input debug sections.

2.135.1 Default

The default is `--no_xrefdbg`.

2.135.2 See also

Reference

- *--list=file* on page 2-77
- *--xref, --no_xref* on page 2-147
- *--xref{from|to}=object(section)* on page 2-149.

2.136 `--xref{from|to}=`*`object(section)`*

This option lists to `stdout` cross-references:

- from input *`section`* in *`object`* to other input sections
- to input *`section`* in *`object`* from other input sections.

This is a useful subset of the listing produced by the `--xref` linker option if you are interested in references from or to a specific input section. You can have multiple occurrences of this option to list references from or to more than one input section.

2.136.1 See also

Reference

- *--list=file* on page 2-77
- *--xref, --no_xref* on page 2-147
- *--xrefdbg, --no_xrefdbg* on page 2-148.

2.137 `--zi_base=`*`address`*

This option specifies the base address of an ER_ZI execution region.

2.137.1 Syntax

`--zi_base=`*`address`*

Where *`address`* must be word-aligned.

2.137.2 Restrictions

The linker ignores `--zi_base` if one of the following options is also specified:

- `--reloc`
- `--rwpi`
- `--split`

You cannot use `--zi_base` with `--scatter`.

2.137.3 See also

Reference

- *--reloc* on page 2-101
- *--rwpi* on page 2-108
- *--scatter=file* on page 2-110
- *--split* on page 2-118

Chapter 3
Linker steering file command reference

The following topics describe the steering file commands supported by the linker, `armlink`:

3.1 EXPORT

The `EXPORT` command specifies that a symbol can be accessed by other shared objects or executables.

Note

A symbol can be exported only if the reference has `STV_DEFAULT` visibility. You must use the `--override_visibility` command-line option to enable the linker to override symbol visibility to `STV_DEFAULT`.

3.1.1 Syntax

```
EXPORT  pattern [AS replacement_pattern] [,pattern [AS replacement_pattern]]
```

where:

`pattern`
: Is a string, optionally including wildcard characters (either * or ?), that matches zero or more defined global symbols. If *`pattern`* does not match any defined global symbol, the linker ignores the command. The operand can match only defined global symbols.

  If the symbol is not defined, the linker issues:

  ```
  Warning: L6331W: No eligible global symbol matches pattern symbol
  ```

`replacement_pattern`
: Is a string, optionally including wildcard characters (either * or ?), to which the defined global symbol is to be renamed. Wild characters must have a corresponding wildcard in *`pattern`*. The characters matched by the *`replacement_pattern`* wildcard are substituted for the *`pattern`* wildcard.

  For example:

  ```
  EXPORT my_func AS func1
  ```

  renames and exports the defined symbol `my_func` as `func1`.

3.1.2 Usage

You cannot export a symbol to a name that already exists. Only one wildcard character (either * or ?) is permitted in `EXPORT`.

The defined global symbol is included in the dynamic symbol table (as *`replacement_pattern`* if given, otherwise as *`pattern`*), if a dynamic symbol table is present.

3.1.3 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

Reference

- *--override_visibility* on page 2-90
- *IMPORT* on page 3-4.

3.2 HIDE

The `HIDE` command makes defined global symbols in the symbol table anonymous.

3.2.1 Syntax

`HIDE  pattern [,pattern]`

where:

pattern	Is a string, optionally including wildcard characters, that matches zero or more defined global symbols. If *pattern* does not match any defined global symbol, the linker ignores the command. You cannot hide undefined symbols.

3.2.2 Usage

`HIDE` and `SHOW` can be used to make certain global symbols anonymous in an output image or partially linked object. Hiding symbols in an object file or library can be useful as a means of protecting intellectual property, as shown in Example 3-1. This example produces a partially linked object with all global symbols hidden, except those beginning with `os_`.

Example 3-1 Using the HIDE command

```
; steer.txt

; Hides all global symbols
HIDE *
; Shows all symbols beginning with 'os_'
SHOW os_*
```

Link this example with the command:

```
armlink --partial input_object.o --edit steer.txt --o partial_object.o
```

You can be link the resulting partial object with other objects, provided they do not contain references to the hidden symbols. When symbols are hidden in the output object, `SHOW` commands in subsequent link steps have no effect on them. The hidden references are removed from the output symbol table.

3.2.3 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

Reference

- *--edit=file_list* on page 2-37
- *--partial* on page 2-94
- *SHOW* on page 3-10.

3.3 IMPORT

The `IMPORT` command specifies that a symbol is defined in a shared object at runtime.

Note

A symbol can be imported only if the reference has `STV_DEFAULT` visibility. You must use the `--override_visibility` command-line option to enable the linker to override symbol visibility to `STV_DEFAULT`.

3.3.1 Syntax

```
IMPORT  pattern [AS replacement_pattern] [,pattern [AS replacement_pattern]]
```

where:

`pattern`
: Is a string, optionally including wildcard characters (either `*` or `?`), that matches zero or more undefined global symbols. If *`pattern`* does not match any undefined global symbol, the linker ignores the command. The operand can match only undefined global symbols.

`replacement_pattern`
: Is a string, optionally including wildcard characters (either `*` or `?`), to which the symbol is to be renamed. Wild characters must have a corresponding wildcard in *`pattern`*. The characters matched by the *`pattern`* wildcard are substituted for the *`replacement_pattern`* wildcard.

For example:

```
IMPORT my_func AS func
```

imports and renames the undefined symbol `my_func` as `func`.

3.3.2 Usage

You cannot import a symbol that has been defined in the current shared object or executable. Only one wildcard character (either `*` or `?`) is permitted in `IMPORT`.

The undefined symbol is included in the dynamic symbol table (as *`replacement_pattern`* if given, otherwise as *`pattern`*), if a dynamic symbol table is present.

Note

The `IMPORT` command only affects undefined global symbols. Symbols that have been resolved by a shared library are implicitly imported into the dynamic symbol table. The linker ignores any `IMPORT` directive that targets an implicitly imported symbol.

3.3.3 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

Reference

- *--override_visibility* on page 2-90
- *EXPORT* on page 3-2.

3.4 RENAME

The RENAME command renames defined and undefined global symbol names.

3.4.1 Syntax

```
RENAME  pattern AS replacement_pattern [,pattern AS replacement_pattern]
```

where:

pattern
: Is a string, optionally including wildcard characters (either * or ?), that matches zero or more global symbols. If *pattern* does not match any global symbol, the linker ignores the command. The operand can match both defined and undefined symbols.

replacement_pattern
: Is a string, optionally including wildcard characters (either * or ?), to which the symbol is to be renamed. Wild characters must have a corresponding wildcard in *pattern*. The characters matched by the *pattern* wildcard are substituted for the *replacement_pattern* wildcard.

For example, for a symbol named `func1`:

```
RENAME f* AS my_f*
```

renames `func1` to `my_func1`.

3.4.2 Usage

You cannot rename a symbol to a global symbol name that already exists, even if the target symbol name is being renamed itself.

You cannot rename a symbol to the same name as another symbol. For example, you cannot do the following:

```
RENAME foo1 bar
RENAME foo2 bar
```

Renames only take effect at the end of the link step. Therefore, renaming a symbol does not remove its original name. This means that you cannot do the following:

```
RENAME func1 func2
RENAME func2 func3
```

The linker gives an error that func1 cannot be renamed to func2 as a symbol already exists with that name.

Only one wildcard character (either * or ?) is permitted in RENAME.

3.4.3 Example

Given an image containing the symbols `func1`, `func2`, and `func3`, you might have a steering file containing the following commands:

```
;invalid, func2 already exists EXPORT func1 AS func2

; valid RENAME func3 AS b2
;invalid, func3 still exists because the link step is not yet complete EXPORT func1 AS
func3
```

3.4.4 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

3.5 REQUIRE

The `REQUIRE` command creates a `DT_NEEDED` tag in the dynamic array. `DT_NEEDED` tags specify dependencies to other shared objects used by the application, for example, a shared library.

3.5.1 Syntax

```
REQUIRE  pattern [,pattern]
```

where:

pattern	Is a string representing a filename. No wild characters are permitted.

3.5.2 Usage

The linker inserts a `DT_NEEDED` tag with the value of *`pattern`* into the dynamic array. This tells the dynamic loader that the file it is currently loading requires *`pattern`* to be loaded.

Note

`DT_NEEDED` tags inserted as a result of a `REQUIRE` command are added after `DT_NEEDED` tags generated from shared objects or *dynamically linked libraries* (DLLs) placed on the command line.

3.5.3 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

3.6 RESOLVE

The RESOLVE command matches specific undefined references to a defined global symbol.

3.6.1 Syntax

RESOLVE *pattern* AS *defined_pattern*

where:

pattern
: Is a string, optionally including wildcard characters (either * or ?), that matches zero or more undefined global symbols. If *pattern* does not match any undefined global symbol, the linker ignores the command. The operand can match only undefined global symbols.

defined_pattern
: Is a string, optionally including wildcard characters, that matches zero or more defined global symbols. If *defined_pattern* does not match any defined global symbol, the linker ignores the command. You cannot match an undefined reference to an undefined symbol.

3.6.2 Usage

RESOLVE is an extension of the existing `armlink --unresolved` command-line option. The difference is that `--unresolved` enables all undefined references to match one single definition, whereas RESOLVE enables more specific matching of references to symbols.

The undefined references are removed from the output symbol table.

RESOLVE works when performing partial-linking and when linking normally.

3.6.3 Example

You might have two files `file1.c` and `file2.c`, as shown in the following example:

Example 3-2 Using the RESOLVE command

```
file1.c

extern int foo;
extern void MP3_Init(void);
extern void MP3_Play(void);

int main(void)
{
  int x = foo + 1;
  MP3_Init();
  MP3_Play();
  return x;
}


file2.c:

int foobar;
void MyMP3_Init()
{
}
```

```
void MyMP3_Play()
{
}
```

Create a steering file, `ed.txt`, containing the line:

```
RESOLVE MP3* AS MyMP3*.
```

Enter the following command:

```
armlink file1.o file2.o --edit ed.txt --unresolved foobar
```

This command has the following effects:

- The references from `file1.o` (`foo`, `MP3_Init()` and `MP3_Play()`) are matched to the definitions in `file2.o` (`foobar`, `MyMP3_Init()` and `MyMP3_Play()` respectively), as specified by the steering file `ed.txt`.
- The `RESOLVE` command in `ed.txt` matches the `MP3` functions and the `--unresolved` option matches any other remaining references, in this case, `foo` to `foobar`.
- The output symbol table, whether it is an image or a partial object, does not contain the symbols `foo`, `MP3_Init` or `MP3_Play`.

3.6.4 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

Reference

- *--edit=file_list* on page 2-37
- *--unresolved=symbol* on page 2-136

3.7 SHOW

The `SHOW` command makes global symbols visible. This command is useful if you want to make a specific symbol visible that is hidden using a `HIDE` command with a wildcard.

3.7.1 Syntax

```
SHOW  pattern [,pattern]
```

where:

`pattern`	Is a string, optionally including wildcard characters, that matches zero or more global symbols. If *`pattern`* does not match any global symbol, the linker ignores the command.

3.7.2 Usage

The usage of `SHOW` is closely related to that of `HIDE`.

3.7.3 See also

Concepts

Using the Linker:

- *What is a steering file?* on page 7-23.

Reference

- *HIDE* on page 3-3.

Chapter 4
Formal syntax of the scatter file

The following topics describe the format of scatter files:

Concepts

- *Symbol related function in a scatter file* on page 4-40
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *BNF notation used in scatter-loading description syntax* on page 4-3
- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Load region attributes* on page 4-7
- *Syntax of an execution region description* on page 4-9
- *Execution region attributes* on page 4-11
- *Address attributes for load and execution regions* on page 4-14
- *Syntax of an input section description* on page 4-22
- *AlignExpr(expr, align) function* on page 4-42
- *GetPageSize() function* on page 4-43
- *SizeOfHeaders() function* on page 4-44.

4.1 BNF notation used in scatter-loading description syntax

Table 4-1 summarizes the *Backus-Naur Form* (BNF) symbols that are used to describe a formal language.

Table 4-1 BNF notation

Symbol	Description
"	Quotation marks are used to indicate that a character that is normally part of the BNF syntax is used as a literal character in the definition. The definition `B"+"C`, for example, can only be replaced by the pattern `B+C`. The definition `B+C` can be replaced by, for example, patterns `BC`, `BBC`, or `BBBC`.
A ::= *B*	Defines *A* as *B*. For example, `A::= B"+" | C` means that *A* is equivalent to either `B+` or `C`. The `::=` notation is used to define a higher level construct in terms of its components. Each component might also have a `::=` definition that defines it in terms of even simpler components. For example, `A::= B` and `B::= C | D` means that the definition *A* is equivalent to the patterns `C` or `D`.
[*A*]	Optional element *A*. For example, `A::= B[C]D` means that the definition `A` can be expanded into either `BD` or `BCD`.
A+	Element *A* can have one or more occurrences. For example, `A::= B+` means that the definition `A` can be expanded into `B`, `BB`, or `BBB`.
*A**	Element *A* can have zero or more occurrences.
A | *B*	Either element *A* or *B* can occur, but not both.
(*A B*)	Element *A* and *B* are grouped together. This is particularly useful when the | operator is used or when a complex pattern is repeated. For example, `A::=(B C)+ (D | E)` means that the definition `A` can be expanded into any of `BCD`, `BCE`, `BCBCD`, `BCBCE`, `BCBCBCD`, or `BCBCBCE`.

4.1.1 See also

Concepts

- *Syntax of a scatter file* on page 4-4.

4.2 Syntax of a scatter file

The following figure shows the components and organization of a typical scatter file:

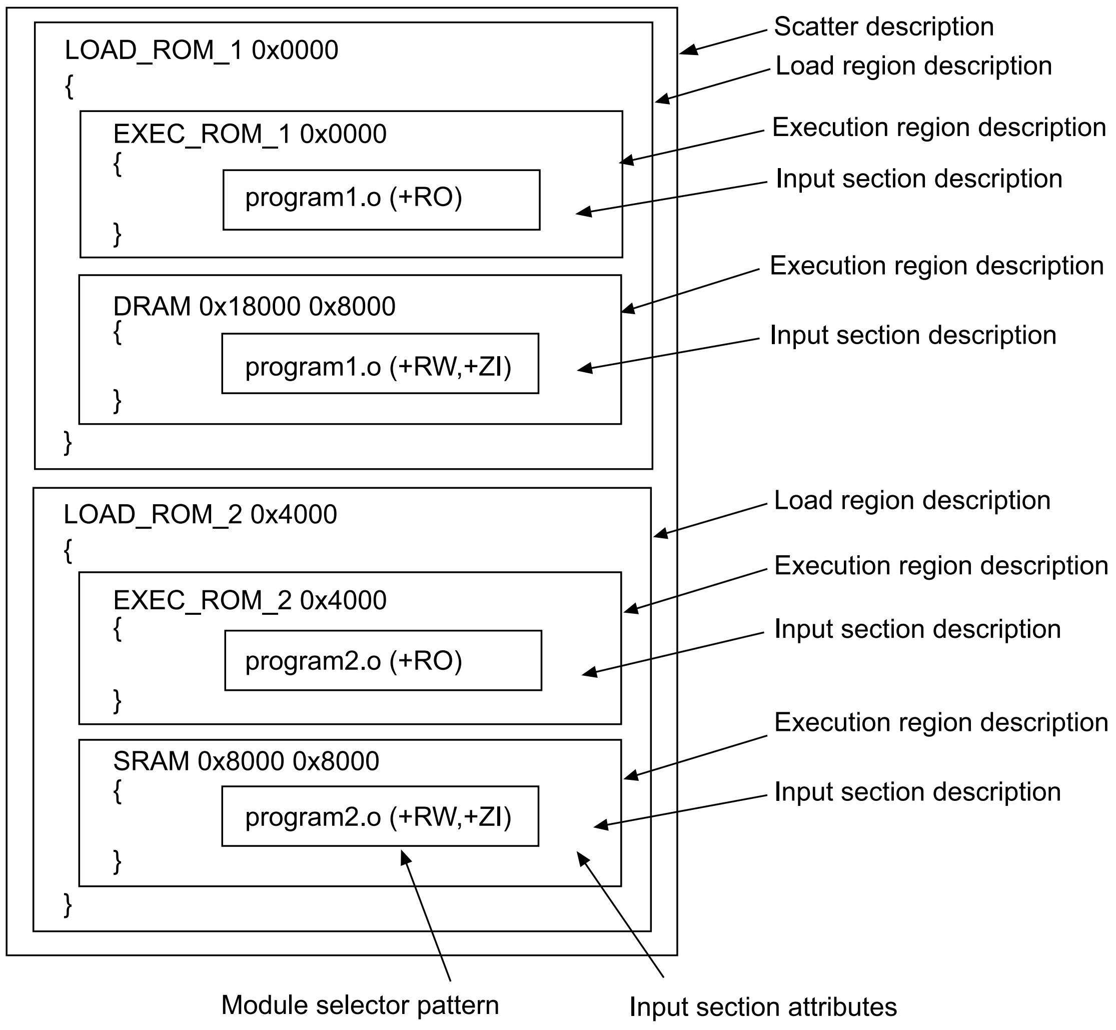


Figure 4-1 Components of a scatter file

4.2.1 See also

Tasks

Using the Linker:

- Chapter 8 *Using scatter files*.

Concepts

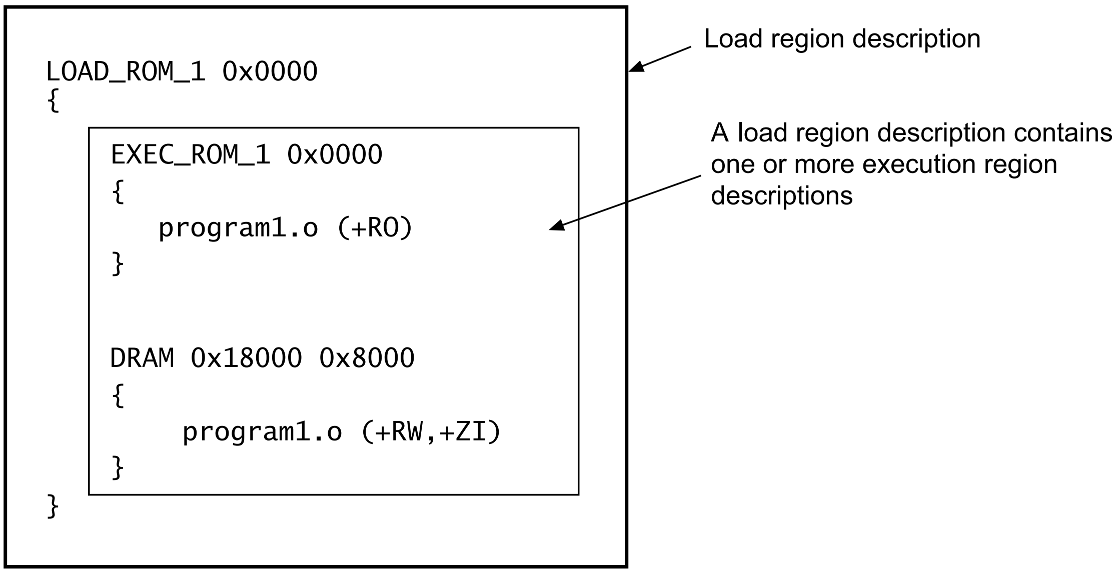
- *About load region descriptions* on page 4-5
- *About execution region descriptions* on page 4-8

4.3 About load region descriptions

A load region description has:

- a name (used by the linker to identify different load regions)
- a base address (the start address for the code and data in the load view)
- attributes that specify the properties of the load region
- an optional maximum size specification
- one or more execution regions.

The following figure shows the components of a typical load region description:



```
LOAD_ROM_1 0x0000
{
    EXEC_ROM_1 0x0000
    {
       program1.o (+RO)
    }

    DRAM 0x18000 0x8000
    {
         program1.o (+RW,+ZI)
    }
}
```

Load region description
A load region description contains one or more execution region descriptions

Figure 4-2 Components of a load region description

4.3.1 See also

Tasks

Using the Linker:

- *About creating regions on page boundaries* on page 8-52
- Chapter 8 *Using scatter files*.

Concepts

- *About Expression evaluation in scatter files* on page 4-30.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Load region attributes* on page 4-7
- *Address attributes for load and execution regions* on page 4-14.

4.4 Syntax of a load region description

The syntax of a load region description, in *Backus-Naur Form* (BNF), is:

```
load_region_description ::=
  load_region_name  (base_address  | ("+" offset)) [attribute_list] [max_size]
       "{"
          execution_region_description+
       "}"
```

where:

`load_region_name`
Names the load region.

`base_address` Specifies the address where objects in the region are to be linked. `base_address` must satisfy the alignment constraints of the load region.

`+offset` Describes a base address that is `offset` bytes beyond the end of the preceding load region. The value of `offset` must be zero modulo four. If this is the first load region, then `+offset` means that the base address begins `offset` bytes from zero.

If you use `+offset`, then the load region might inherit certain attributes from a previous load region.

`attribute_list`

The attributes that specify the properties of the load region contents.

`max_size` Specifies the maximum size of the load region. This is the size of the load region before any decompression or zero initialization take place. If the optional `max_size` value is specified, `armlink` generates an error if the region has more than `max_size` bytes allocated to it.

`execution_region_description`

Specifies the execution region name, address, and contents.

——— **Note** ———

The *Backus-Naur Form* (BNF) definitions contain additional line returns and spaces to improve readability. They are not required in the scatter-loading definition and are ignored if present in the file.

———————————

4.4.1 See also

Concepts

- *About load region descriptions* on page 4-5
- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Inheritance rules for load region address attributes* on page 4-18
- *About Expression evaluation in scatter files* on page 4-30.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description*
- *Load region attributes* on page 4-7
- *Address attributes for load and execution regions* on page 4-14.

4.5 Load region attributes

The load region attributes are:

`ABSOLUTE`	Absolute address. The load address of the region is specified by the base designator. This is the default, unless you use `PI` or `RELOC`.
`ALIGN` *`alignment`*	Increase the alignment constraint for the load region from 4 to *`alignment`*. *`alignment`* must be a positive power of 2. If the load region has a *`base_address`* then this must be *`alignment`* aligned. If the load region has a *`+offset`* then the linker aligns the calculated base address of the region to an *`alignment`* boundary. This can also affect the offset in the ELF file. For example, the following causes the data for `FOO` to be written out at 4k offset into the ELF file: `FOO +4 ALIGN 4096`
`NOCOMPRESS`	RW data compression is enabled by default. The `NOCOMPRESS` keyword enables you to specify that the contents of a load region must not be compressed in the final image.
`OVERLAY`	The `OVERLAY` keyword enables you to have multiple load regions at the same address. ARM tools do not provide an overlay mechanism. To use multiple load regions at the same address, you must provide your own overlay manager.
`PI`	This region is position independent.
`PROTECTED`	The `PROTECTED` keyword prevents: • overlapping of load regions • veneer sharing • string sharing with the `--merge` option.
`RELOC`	This region is relocatable.

4.5.1 See also

Concepts

- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Inheritance rules for the RELOC address attribute* on page 4-20.

Using the Linker:

- *About load region descriptions* on page 4-5
- *Section alignment with the linker* on page 4-22
- *Veneer sharing* on page 4-27
- *Generation of position independent to absolute veneers* on page 4-29
- *Reuse of veneers when scatter-loading* on page 4-30
- *Optimization with RW data compression* on page 5-12
- *Placement of sections with overlays* on page 8-42
- *About creating regions on page boundaries* on page 8-52.

Reference

- *--merge, --no_merge* on page 2-87
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

4.6 About execution region descriptions

An execution region description has:

- a name (used by the linker to identify different execution regions)
- a base address (either absolute or relative)
- attributes that specify the properties of the execution region
- an optional maximum size specification
- one or more input section descriptions (the modules placed into this execution region).

The following figure shows the components of a typical execution region description:

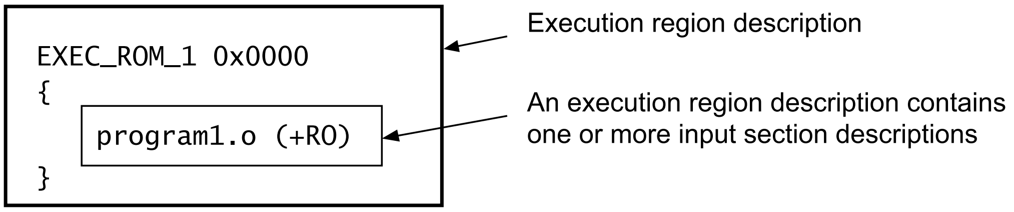

EXEC_ROM_1 0x0000
{
program1.o (+RO)
}
Execution region description
An execution region description contains one or more input section descriptions

Figure 4-3 Components of an execution region description

4.6.1 See also

Tasks

Using the Linker:

- Chapter 8 *Using scatter files*.

Concepts

- *About Expression evaluation in scatter files* on page 4-30.

Using the Linker:

- *Placement of sections with overlays* on page 8-42
- *About creating regions on page boundaries* on page 8-52.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of an execution region description* on page 4-9
- *Execution region attributes* on page 4-11
- *Address attributes for load and execution regions* on page 4-14
- *About input section descriptions* on page 4-21.

4.7 Syntax of an execution region description

The syntax of an execution region description, in *Backus-Naur Form* (BNF), is:

```
execution_region_description ::=

  exec_region_name (base_address | "+" offset) [attribute_list] [max_size | length]
        "{"
            input_section_description*
        "}"
```

where:

`exec_region_name`

Names the execution region.

`base_address` Specifies the address where objects in the region are to be linked. `base_address` must be word-aligned.

Note

Using `ALIGN` on an execution region causes both the load address and execution address to be aligned.

`+offset` Describes a base address that is `offset` bytes beyond the end of the preceding execution region. The value of `offset` must be zero modulo four.

If this is the first execution region in the load region then `+offset` means that the base address begins `offset` bytes after the base of the containing load region.

If you use `+offset`, then the execution region might inherit certain attributes from the parent load region, or from a previous execution region within the same load region.

`attribute_list`

The attributes that specify the properties of the execution region contents.

`max_size` For an execution region marked `EMPTY` or `FILL` the `max_size` value is interpreted as the length of the region. Otherwise the `max_size` value is interpreted as the maximum size of the execution region.

`[-]length` Can only be used with `EMPTY` to represent a stack that grows down in memory. If the length is given as a negative value, the `base_address` is taken to be the end address of the region.

`input_section_description`

Specifies the content of the input sections.

Note

The *Backus-Naur Form* (BNF) definitions contain additional line returns and spaces to improve readability. They are not required in the scatter-loading definition and are ignored if present in the file.

4.7.1 See also

Tasks

Using the Linker:

- Chapter 8 *Using scatter files*.

Concepts

- *About execution region descriptions* on page 4-8
- *Considerations when using a relative address +offset for execution regions* on page 4-17
- *About Expression evaluation in scatter files* on page 4-30.

Using the Linker:

- *Placement of sections with overlays* on page 8-42
- *About creating regions on page boundaries* on page 8-52.

Reference

- *Syntax of a scatter file* on page 4-4
- *Execution region attributes* on page 4-11
- *Address attributes for load and execution regions* on page 4-14
- *Inheritance rules for load region address attributes* on page 4-18
- *Inheritance rules for execution region address attributes* on page 4-19
- *Inheritance rules for the RELOC address attribute* on page 4-20
- *About input section descriptions* on page 4-21.

4.8 Execution region attributes

The execution region attributes are:

`ABSOLUTE`
: Absolute address. The execution address of the region is specified by the base designator.

`ALIGN` *`alignment`*
: Increase the alignment constraint for the execution region from 4 to *`alignment`*. *`alignment`* must be a positive power of 2. If the execution region has a *`base_address`* then this must be *`alignment`* aligned. If the execution region has a *`+offset`* then the linker aligns the calculated base address of the region to an *`alignment`* boundary.

  ——— Note ———

  `ALIGN` on an execution region causes both the load address and execution address to be aligned. This can result in padding being added to the ELF file. To align only the execution address, use the `AlignExpr` expression on the base address.

  ————————

`ALIGNALL` *`value`*
: Increases the alignment of sections within the execution region.

  The value must be a positive power of 2 and must be greater than or equal to 4.

`ANY_SIZE` *`max_size`*
: Specifies the maximum size within the execution region that `armlink` can fill with unassigned sections. You can use a simple expression to specify the *`max_size`*. That is, you cannot use functions such as `ImageLimit()`.

  ——— Note ———

  `max_size` is not the contingency, but the maximum size permitted for placing unassigned sections in an execution region. For example, if an execution region is to be filled only with `.ANY` sections, a two percent contingency is still set aside for veneers. This leaves 98% of the region for `.ANY` section assignements.

  ————————

  Be aware of the following restrictions when using this keyword:

  - *`max_size`* must be less than or equal to the region size
  - you can use `ANY_SIZE` on a region without a `.ANY` selector but it is ignored by `armlink`.

`EMPTY [-]`*`length`*
: Reserves an empty block of memory of a given *`size`* in the execution region, typically used by a heap or stack. No section can be placed in a region with the `EMPTY` attribute.

  `length` represent a stack that grows down in memory. If the length is given as a negative value, the *`base_address`* is taken to be the end address of the region.

`FILL` *`value`*
: Creates a linker generated region containing a *`value`*. If you specify `FILL`, you must give a value, for example: `FILL 0xFFFFFFFF`. The `FILL` attribute replaces the following combination: `EMPTY ZEROPAD PADVALUE`.

  In certain situations, for example, simulation, this is preferable to spending a long time in a zeroing loop.

`FIXED`
: Fixed address. The linker attempts to make the execution address equal the load address. This makes the region a root region. If this is not possible the linker produces an error.

Note

The linker inserts padding with this attribute.

`NOCOMPRESS` RW data compression is enabled by default. The `NOCOMPRESS` keyword enables you to specify that RW data in an execution region must not be compressed in the final image.

`OVERLAY` Use for sections with overlaying address ranges. If consecutive execution regions have the same `+`*`offset`* then they are given the same base address.

`PADVALUE` Defines the value of any padding. If you specify `PADVALUE`, you must give a value, for example:

```
EXEC 0x10000 PADVALUE 0xFFFFFFFF EMPTY ZEROPAD 0x2000
```

This creates a region of size `0x2000` full of 0xFFFFFFFF.

`PADVALUE` must be a word in size. `PADVALUE` attributes on load regions are ignored.

`PI` This region contains only position independent sections.

`SORTTYPE` Specifies the sorting algorithm for the execution region, for example:

```
ER1 +0 SORTTYPE CallTree
```

`UNINIT` Use to create execution regions containing uninitialized data or memory-mapped I/O.

`ZEROPAD` Zero-initialized sections are written in the ELF file as a block of zeros and, therefore, do not have to be zero-filled at runtime.

This sets the load length of a ZI output section to `Image$$`*`region_name`*`$$ZI$$Length`.

Only root execution regions can be zero-initialized using the `ZEROPAD` attribute. Using the `ZEROPAD` attribute with a non root execution region generates a warning and the attribute is ignored.

In certain situations, for example, simulation, this is preferable to spending a long time in a zeroing loop.

4.8.1 See also

Concepts

- *About execution region descriptions* on page 4-8
- *Considerations when using a relative address +offset for execution regions* on page 4-17
- *Behavior when .ANY sections overflow because of linker-generated content* on page 4-28
- *About Expression evaluation in scatter files* on page 4-30.

Using the Linker:

- *Section alignment with the linker* on page 4-22
- *Optimization with RW data compression* on page 5-12
- *Image$$ execution region symbols* on page 7-6
- *Load$$ execution region symbols* on page 7-7
- *Placement of sections with overlays* on page 8-42
- *About creating regions on page boundaries* on page 8-52
- *Overalignment of execution regions and input sections* on page 8-54
- *Using expression evaluation in a scatter file to avoid padding* on page 8-57.

Reference

- *--any_contingency* on page 2-5
- *--sort=algorithm* on page 2-116
- *Syntax of a scatter file* on page 4-4
- *Syntax of an execution region description* on page 4-9
- *Syntax of an input section description* on page 4-22
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41
- *AlignExpr(expr, align) function* on page 4-42.

4.9 Address attributes for load and execution regions

A subset of the load and execution region attributes inform the linker about the content of the region and how it behaves after linking. These attributes are:

`ABSOLUTE`	The content is placed at a fixed address that does not change after linking.
`PI`	The content does not depend on any fixed address and might be moved after linking without any extra processing.
`RELOC`	The content depends on fixed addresses, relocation information is output to enable the content to be moved to another location by another tool. **Note** You cannot explicitly use this attribute for an execution region.
`OVERLAY`	The content is placed at a fixed address that does not change after linking. The content might overlap with other regions designated as `OVERLAY` regions.

4.9.1 Inheritance rules for address attributes

In general, all the execution regions within a load region have the same address attribute. To make this easy to select, the address attributes can be inherited from a previous region so that they only have to be set in one place. The rules for setting and inheriting address attributes are:

- Explicitly setting the address attribute:
  - A load region can be explicitly set with the `ABSOLUTE`, `PI`, `RELOC`, or `OVERLAY` attributes.
  - An execution region can be explicitly set with the `ABSOLUTE`, `PI`, or `OVERLAY` attributes. An execution region can only inherit the `RELOC` attribute from the parent load region.
- Implicitly setting the address attribute when none is specified:
  - The `OVERLAY` attribute cannot be inherited. A region with the `OVERLAY` attribute cannot inherit.
  - A base address load or execution region always defaults to `ABSOLUTE`.
  - A *`+offset`* load region inherits the address attribute from the previous load region or `ABSOLUTE` if no previous load region exists.
  - A *`+offset`* execution region inherits the address attribute from the previous execution region or parent load region if no previous execution region exists.

4.9.2 See also

Concepts

- *About load region descriptions* on page 4-5
- *About execution region descriptions* on page 4-8
- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Considerations when using a relative address +offset for execution regions* on page 4-17

Using the Linker:

- *Placement of sections with overlays* on page 8-42.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Load region attributes* on page 4-7

- *Syntax of an execution region description* on page 4-9
- *Execution region attributes* on page 4-11.

4.10 Considerations when using a relative address +offset for load regions

Be aware of the following when using `+offset` to specify a load region base address:

- If the `+offset` load region LR2 follows a load region LR1 containing ZI data, then LR2 overlaps the ZI data. To fix this, use the `ImageLimit()` function to specify the base address of LR2.
- A `+offset` load region LR2 inherits the attributes of the load region LR1 immediately before it, unless:
  - — LR1 has the `OVERLAY` attribute
  - — LR2 has an explicit attribute set.

  If a load region is unable to inherit an attribute, then it gets the attribute `ABSOLUTE`.

4.10.1 See also

Concepts

- *Inheritance rules for load region address attributes* on page 4-18
- *Execution address built-in functions for use in scatter files* on page 4-34.

4.11 Considerations when using a relative address +offset for execution regions

Be aware of the following when using `+offset` to specify an execution region base address:

- The first execution region inherits the attributes of the parent load region, unless an attribute is explicitly set on that execution region.
- A `+offset` execution region ER2 inherits the attributes of the execution region ER1 immediately before it, unless:
  - ER1 has the `OVERLAY` attribute
  - ER2 has an explicit attribute set.

  If an execution region is unable to inherit an attribute, then it gets the attribute `ABSOLUTE`.
- If the parent load region has the `RELOC` attribute, then all execution regions within that load region must have a `+offset` base address.

4.11.1 See also

Concepts

- *Inheritance rules for execution region address attributes* on page 4-19
- *Inheritance rules for the RELOC address attribute* on page 4-20.

4.12 Inheritance rules for load region address attributes

For a load region to inherit the attributes of a previous load region, specify a `+offset` base address for that region. A load region cannot inherit attributes if:

- you explicitly set the attribute of that load region
- the load region immediately before has the `OVERLAY` attribute.

You can explicitly set a load region with the `ABSOLUTE`, `PI`, `RELOC`, or `OVERLAY` address attributes.

This example shows the inheritance rules for setting the address attributes of load regions:

Example 4-1 Load region inheritance

```
LR1 0x8000 PI
{
    ...
}
LR2 +0            ; LR2 inherits PI from LR1
{
    ...
}
LR3 0x1000        ; LR3 does not inherit because it has no relative base
                    address, gets default of ABSOLUTE
{
    ...
}
LR4 +0            ; LR4 inherits ABSOLUTE from LR3
{
    ...
}
LR5 +0 RELOC      ; LR5 does not inherit because it explicitly sets RELOC
{
    ...
}
LR6 +0 OVERLAY    ; LR6 does not inherit, an OVERLAY cannot inherit
{
    ...
}
LR7 +0            ; LR7 cannot inherit OVERLAY, gets default of ABSOLUTE
{
    ...
}
```

4.12.1 See also

Concepts

- *Address attributes for load and execution regions* on page 4-14
- *Considerations when using a relative address +offset for load regions* on page 4-16.

4.13 Inheritance rules for execution region address attributes

For an execution region to inherit the attributes of a previous execution region, specify a `+offset` base address for that region. The first `+offset` execution region can inherit the attributes of the parent load region. An execution region cannot inherit attributes if you:

- explicitly set the attribute of that execution region
- the previous execution region has the `OVERLAY` attribute.

You can explicitly set an execution region with the `ABSOLUTE`, `PI`, or `OVERLAY` attributes. However, an execution region can only inherit the `RELOC` attribute from the parent load region.

This example shows the inheritance rules for setting the address attributes of execution regions:

Example 4-2 Execution region inheritance

```
LR1 0x8000 PI
{
    ER1 +0        ; ER1 inherits PI from LR1
    {
        ...
    }
    ER2 +0        ; ER2 inherits PI from ER1
    {
        ...
    }
    ER3 0x10000   ; ER3 does not inherit because it has no relative base
                    address and gets the default of ABSOLUTE
    {
        ...
    }
    ER4 +0        ; ER4 inherits ABSOLUTE from ER3
    {
        ...
    }
    ER5 +0 PI     ; ER5 does not inherit, it explicitly sets PI
    {
        ...
    }
    ER6 +0 OVERLAY ; ER6 does not inherit, an OVERLAY cannot inherit
    {
        ...
    }
    ER7 +0       ; ER7 cannot inherit OVERLAY, gets the default of ABSOLUTE
    {
        ...
    }
}
```

4.13.1 See also

Concepts

- *Address attributes for load and execution regions* on page 4-14
- *Considerations when using a relative address +offset for execution regions* on page 4-17.

4.14 Inheritance rules for the RELOC address attribute

You can explicitly set the RELOC attribute for a load region. However, an execution region can only inherit the RELOC attribute from the parent load region.

Note

For a Base Platform linking model, if a load region has the RELOC attribute, then all execution regions within that load region must have a *+offset* base address. This ensures the execution regions inherit the relocations from the parent load region.

This example shows the inheritance rules for setting the address attributes with RELOC:

Example 4-3 Inheriting RELOC

```
LR1 0x8000 RELOC
{
    ER1 +0 ; inherits RELOC from LR1
    {
        ...
    }
    ER2 +0 ; inherits RELOC from ER1
    {
        ...
    }
    ER3 +0 RELOC ; Error cannot explicitly set RELOC on an execution region
    {
        ...
    }
}
```

4.14.1 See also

Concepts

- *Address attributes for load and execution regions* on page 4-14
- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Considerations when using a relative address +offset for execution regions* on page 4-17.

4.15 About input section descriptions

An input section description is a pattern that identifies input sections by:

- Module name (object filename, library member name, or library filename). The module name can use wildcard characters.
- Input section name, or input section attributes such as `READ-ONLY`, or `CODE`. You can use wildcard characters for the input section name.
- Symbol name.

The following figure shows the components of a typical input section description.

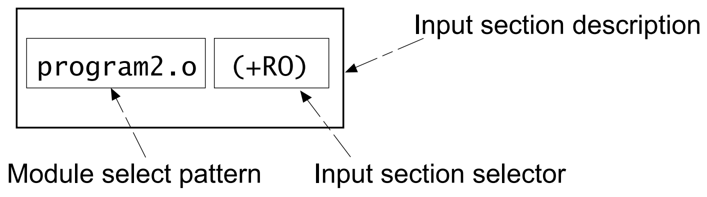


Figure 4-4 Components of an input section description

Note

Ordering in an execution region does not affect the ordering of sections in the output image.

4.15.1 See also

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of an input section description* on page 4-22.

4.16 Syntax of an input section description

The syntax of an input section description, in *Backus-Naur Form* (BNF), is:

```
input_section_description ::=

  module_select_pattern
       [ "(" input_section_selector ( "," input_section_selector )* ")" ]
input_section_selector ::=

       ("+" input_section_attr | input_section_pattern | input_symbol_pattern |
section_properties)
```

where:

module_select_pattern

A pattern constructed from literal text. The wildcard character `*` matches zero or more characters and ? matches any single character.

Matching is case-insensitive, even on hosts with case-sensitive file naming.

Use `*.o` to match all objects. Use `*` to match all object files and libraries.

An input section matches a module selector pattern when *module_select_pattern* matches one of the following:

- The name of the object file containing the section.
- The name of the library member (without leading path name).
- The full name of the library (including path name) the section is extracted from. If the names contain spaces, use wild characters to simplify searching. For example, use `*libname.lib` to match `C:\lib dir\libname.lib`.

The following module selector patterns describe the placement order of an input section within the execution region:

`.ANY` module selector for unassigned sections

The special module selector pattern `.ANY` enables you to assign input sections to execution regions without considering their parent module. Use `.ANY` to fill up the execution regions with input sections that do not have to be placed at specific locations.

Modified selectors

You cannot have two `*` selectors in a scatter file. You can, however, use two modified selectors, for example `*A` and `*B`, and you can use a `.ANY` selector together with a `*` module selector. The `*` module selector has higher precedence than `.ANY`. If the portion of the file containing the `*` selector is removed, the `.ANY` selector then becomes active.

Note

- Only input sections that match both *module_select_pattern* and at least one *input_section_attr* or *input_section_pattern* are included in the execution region.

  If you omit (+ *input_section_attr)* and *(input_section_pattern)*, the default is `+RO`.
- Do not rely on input section names generated by the compiler, or used by ARM library code. These can change between compilations if, for example, different compiler options are used. In addition, section naming conventions used by the compiler are not guaranteed to remain constant between releases.

`input_section_attr`

An attribute selector matched against the input section attributes. Each `input_section_attr` follows a `+`.

If you are specifying a pattern to match the input section name, the name must be preceded by a `+`. You can omit any comma immediately followed by a `+`.

The selectors are not case-sensitive. The following selectors are recognized:

- `RO-CODE`
- `RO-DATA`
- `RO`, selects both `RO-CODE` and `RO-DATA`
- `RW-DATA`
- `RW-CODE`
- `RW`, selects both `RW-CODE` and `RW-DATA`
- `ZI`
- `ENTRY`, that is, a section containing an `ENTRY` point.

The following synonyms are recognized:

- `CODE` for `RO-CODE`
- `CONST` for `RO-DATA`
- `TEXT` for `RO`
- `DATA` for `RW`
- `BSS` for `ZI`.

The following pseudo-attributes are recognized:

- `FIRST`
- `LAST`.

Use `FIRST` and `LAST` to mark the first and last sections in an execution region if the placement order is important. For example, if a specific input section must be first in the region and an input section containing a checksum must be last.

There can be only one `FIRST` or one `LAST` attribute for an execution region, and it must follow a single `input_section_attr`. For example:

`*(section, +FIRST)`

This pattern is correct.

`*(+FIRST, section)`

This pattern is incorrect and produces an error message.

`input_section_pattern`

A pattern that is matched, without case sensitivity, against the input section name. It is constructed from literal text. The wildcard character `*` matches 0 or more characters, and ? matches any single character.

Note

If you use more than one `input_section_pattern`, ensure that there are no duplicate patterns in different execution regions to avoid ambiguity errors.

`input_symbol_pattern`

You can select the input section by the name of a global symbol that the section defines. This enables you to choose individual sections with the same name from partially linked objects.

The `:gdef:` prefix distinguishes a global symbol pattern from a section pattern. For example, use `:gdef:mysym` to select the section that defines `mysym`. The following example shows a description file in which `ExecReg1` contains the section that defines global symbol `mysym1`, and the section that contains global symbol `mysym2`:

```
LoadRegion 0x8000
{
    ExecReg1 +0
    {
        *(:gdef:mysym1)
        *(:gdef:mysym2)
    }
                        ; rest of scatter-loading description
}
```

——— **Note** ———

If you use more than one *`input_symbol_pattern`*, ensure that there are no duplicate patterns in different execution regions to avoid ambiguity errors.

The order of input section descriptors is not significant.

———

`section_properties`

A section property can be `+FIRST`, `+LAST`, and `OVERALIGN` *`value`*.

The value for `OVERALIGN` must be a positive power of 2 and must be greater than or equal to 4.

——— **Note** ———

The BNF definitions contain additional line returns and spaces to improve readability. They are not required in the scatter-loading definition and are ignored if present in the file.

———

4.16.1 Examples of module select patterns

Examples of *`module_select_pattern`* specifications are:

- `*` matches any module or library
- `*.o` matches any object module
- `math.o` matches the `math.o` module
- `*armlib*` matches all C libraries supplied by ARM
- `*math.lib` matches any library path ending with `math.lib`. For example, `C:\apps\lib\math\satmath.lib`.

4.16.2 Examples of input section selector patterns

Examples of *`input_section_selector`* specifications are:

- `+RO` is an input section attribute that matches all RO code and all RO data
- `+RW,+ZI` is an input section attribute that matches all RW code, all RW data, and all ZI data
- `BLOCK_42` is an input section pattern that matches sections named `BLOCK_42`. There can be multiple ELF sections with the same `BLOCK_42` name that possess different attributes, for example `+RO-CODE,+RW`.

4.16.3 See also

Tasks

- *Behavior when .ANY sections overflow because of linker-generated content* on page 4-28.

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

- *About input section descriptions* on page 4-21.

Using the Linker:

- *Examples of using placement algorithms for .ANY sections* on page 8-26
- *Example of next_fit algorithm showing behavior of full regions, selectors, and priority* on page 8-28
- *Examples of using sorting algorithms for .ANY sections* on page 8-30.

Using the Linker:

- *Overalignment of execution regions and input sections* on page 8-54.

Reference

- *Syntax of a scatter file* on page 4-4.

4.17 How the linker resolves multiple matches when processing scatter files

An input section must be unique. In the case of multiple matches, the linker attempts to assign the input section to a region based on a `module_select_pattern` and `input_section_selector` pair that is the most specific. However, if a unique match cannot be found, the linker faults the scatter-loading description.

The following variables are used to describe how the linker matches multiple input sections:

- *m1* and *m2* represent module selector patterns
- *s1* and *s2* represent input section selectors.

For example, if input section A matches *m1,s1* for execution region R1, and A matches *m2,s2* for execution region R2, the linker:

- assigns A to R1 if *m1,s1* is more specific than *m2,s2*
- assigns A to R2 if *m2,s2* is more specific than *m1,s1*
- diagnoses the scatter-loading description as faulty if *m1,s1* is not more specific than *m2,s2* and *m2,s2* is not more specific than *m1,s1*.

`armlink` uses the following sequence to determine the most specific `module_select_pattern`, `input_section_selector` pair:

1. For the module selector patterns:

   m1 is more specific than *m2* if the text string *m1* matches pattern *m2* and the text string *m2* does not match pattern *m1*.

2. For the input section selectors:
   - If *s1* and *s2* are both patterns matching section names, the same definition as for module selector patterns is used.
   - If one of *s1*, *s2* matches the input section name and the other matches the input section attributes, *s1* and *s2* are unordered and the description is diagnosed as faulty.
   - If both *s1* and *s2* match input section attributes, the determination of whether *s1* is more specific than *s2* is defined by the relationships below:
     - — `ENTRY` is more specific than `RO-CODE`, `RO-DATA`, `RW-CODE` or `RW-DATA`
     - — `RO-CODE` is more specific than `RO`
     - — `RO-DATA` is more specific than `RO`
     - — `RW-CODE` is more specific than `RW`
     - — `RW-DATA` is more specific than `RW`
     - — There are no other members of the (*s1* more specific than *s2*) relationship between section attributes.

3. For the `module_select_pattern, input_section_selector` pair, *m1,s1* is more specific than *m2,s2* only if any of the following are true:

   a. *s1* is a literal input section name that is, it contains no pattern characters, and *s2* matches input section attributes other than `+ENTRY`

   b. *m1* is more specific than *m2*

   c. *s1* is more specific than *s2*.

   The conditions are tested in order so condition a takes precedence over condition b and c, and condition b takes precedence over condition c.

This matching strategy has the following consequences:

- Descriptions do not depend on the order they are written in the file.

- Generally, the more specific the description of an object, the more specific the description of the input sections it contains.
- The *`input_section_selector`*s are not examined unless:
  - — Object selection is inconclusive.
  - — One selector fully names an input section and the other selects by attribute. In this case, the explicit input section name is more specific than any attribute, other than `ENTRY`, that selects exactly one input section from one object. This is true even if the object selector associated with the input section name is less specific than that of the attribute.

The `.ANY` module selector is available to assign any sections that cannot be resolved from the scatter file.

The following example shows multiple execution regions and pattern matching:

Example 4-4 Multiple execution regions and pattern matching

```
LR_1 0x040000
{
    ER_ROM 0x040000             ; The startup exec region address is the same
    {                           ; as the load address.
        application.o (+ENTRY)  ; The section containing the entry point from
    }                           ; the object is placed here.
    ER_RAM1 0x048000
    {
        application.o (+RO-CODE) ; Other RO code from the object goes here
    }
    ER_RAM2 0x050000
    {
        application.o (+RO-DATA) ; The RO data goes here
    }
    ER_RAM3 0x060000
    {
        application.o (+RW)     ; RW code and data go here
    }
    ER_RAM4 +0                  ; Follows on from end of ER_R3
    {
        *.o (+RO, +RW, +ZI)     ; Everything except for application.o goes here
    }
}
```

4.17.1 See also

Tasks

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Concepts

- *About input section descriptions* on page 4-21.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of an input section description* on page 4-22.

4.18 Behavior when .ANY sections overflow because of linker-generated content

Linker-generated content might cause `.ANY` regions to overflow. This is because the linker does not know the address of a section until it is assigned to a region. Therefore, when filling `.ANY` regions, the linker cannot calculate the contingency space and cannot determine if calling functions require veneers. The linker provides a contingency algorithm that gives a worst-case estimate for padding and an additional two percent for veneers. To enable this algorithm use the `--any_contingency` command-line option.

The following diagram is a representation of the notional image layout during `.ANY` placement:

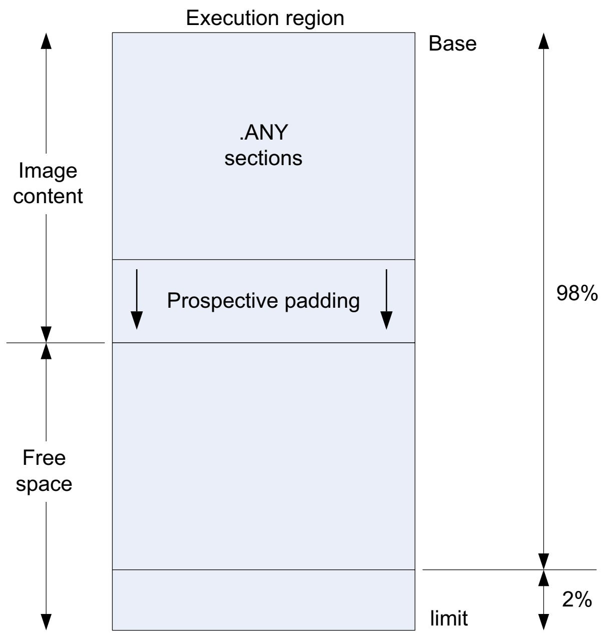


Figure 4-5 .ANY contingency

The downward arrows for prospective padding show that the prospective padding continues to grow as more sections are added to the `.ANY` selector.

Prospective padding is dealt with before the two percent veneer contingency.

When the prospective padding is cleared the priority is set to zero. When the two percent is cleared the priority is decremented again.

You can also use the `ANY_SIZE` keyword on an execution region to specify the maximum amount of space in the region to set aside for `.ANY` section assignments.

4.18.1 See also

Concepts

- *How the linker resolves multiple matches when processing scatter files* on page 4-26.

Using the Linker:

- *Placing unassigned sections with the .ANY module selector* on page 8-23.

Reference

- *--any_contingency* on page 2-5
- *Execution region attributes* on page 4-11
- *Syntax of an input section description* on page 4-22.

4.19 How the linker resolves path names when processing scatter files

The linker matches wildcard patterns in scatter files against any combination of forward slashes and backslashes it finds in path names. This might be useful where the paths are taken from environment variables or multiple sources, or where you want to use the same scatter file to build on Windows or Unix platforms.

Note

Use forward slashes in path names to ensure they are understood on Windows and Unix platforms.

4.19.1 See also

Reference

- *Syntax of a scatter file* on page 4-4.

4.20 About Expression evaluation in scatter files

Scatter files frequently contain numeric constants.You can use specify numeric constants using:

- Expressions.
- Execution address built-in functions.
- `ScatterAssert` function with load address related functions that take an expression as a parameter. An error message is generated if this expression does not evaluate to true.
- The symbol related function, `defined(`*`global_symbol_name`*`) ?` *`expr1`* `:` *`expr2`*.

4.20.1 See also

Concepts

- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Expression usage in scatter files* on page 4-31
- *Expression rules in scatter files* on page 4-32
- *Execution address built-in functions for use in scatter files* on page 4-34
- *ScatterAssert function and load address related functions* on page 4-38
- *Symbol related function in a scatter file* on page 4-40.

4.21 Expression usage in scatter files

Expressions can be used in the following places:

- load and execution region *base_address*
- load and execution region +*offset*
- load and execution region *max_size*
- parameter for the ALIGN, FILL or PADVALUE keywords
- parameter for the ScatterAssert function.

Example 4-5 Specifying the maximum size in terms of an expression

```
LR1 0x8000 (2 * 1024)
{
    ER1 +0 (1 * 1024)
    {
        *(+RO)
    }
    ER2 +0 (1 * 1024)
    {
        *(+RW +ZI)
    }
}
```

4.21.1 See also

Concepts

- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Considerations when using a relative address +offset for execution regions* on page 4-17
- *About Expression evaluation in scatter files* on page 4-30
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression rules in scatter files* on page 4-32
- *Execution address built-in functions for use in scatter files* on page 4-34
- *ScatterAssert function and load address related functions* on page 4-38
- *Symbol related function in a scatter file* on page 4-40.

4.22 Expression rules in scatter files

Expressions follow the C-Precedence rules and are made up of the following:

- Decimal or hexadecimal numbers.
- Arithmetic operators: `+`, `-`, `/`, `*`, `~`, `OR`, and `AND`

  The `OR` and `AND` operators map to the C operators | and & respectively.
- Logical operators: `LOR`, `LAND`, and `!`

  The `LOR` and `LAND` operators map to the C operators `||` and && respectively.
- Relational operators: <, <=, >, >=, and ==

  Zero is returned when the expression evaluates to false and nonzero is returned when true.
- Conditional operator: *`Expression`* `?` *`Expression1`* `:` *`Expression2`*

  This matches the C conditional operator. If *`Expression`* evaluates to nonzero then *`Expression1`* is evaluated otherwise *`Expression2`* is evaluated.

  Note

  When using a conditional operator in a `+`*`offset`* context on an execution region or load region description, the final expression is considered relative only if both *`Expression1`* and *`Expression2`*, are considered relative. For example:

  ```
  er1 0x8000
  {
      ...
  }
  er2 ((ImageLimit(er1) < 0x9000) ? +0 : +0x1000)    ; er2 has a relative address
  {
      ...
  }
  er3 ((ImageLimit(er2) < 0x10000) ? 0x0 : +0)       ; er3 has an absolute address
  {
      ...
  }
  ```
- Functions that return numbers.

All operators match their C counterparts in meaning and precedence.

Expressions are not case sensitive and you can use parentheses for clarity.

4.22.1 See also

Concepts

- *About Expression evaluation in scatter files* on page 4-30
- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Considerations when using a relative address +offset for execution regions* on page 4-17
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression usage in scatter files* on page 4-31

- *Execution address built-in functions for use in scatter files* on page 4-34
- *ScatterAssert function and load address related functions* on page 4-38
- *Symbol related function in a scatter file* on page 4-40.

4.23 Execution address built-in functions for use in scatter files

The execution address related functions can only be used when specifying a *base_address*, +*offset* value, or *max_size*. They map to combinations of the linker defined symbols shown in Table 4-2.

Table 4-2 Execution address related functions

Function	Linker defined symbol value
`ImageBase(`*region_name*`)`	`Image$$`*region_name*`$$Base`
`ImageLength(`*region_name*`)`	`Image$$`*region_name*`$$Length +` `Image$$`*region_name*`$$ZI$$Length`
`ImageLimit(`*region_name*`)`	`Image$$`*region_name*`$$Base +` `Image$$`*region_name*`$$Length +` `Image$$`*region_name*`$$ZI$$Length`

The parameter *region_name* can be either a load or an execution region name. Forward references are not permitted. The *region_name* can only refer to load or execution regions that have already been defined.

Note

You cannot use these functions when using the `.ANY` selector pattern. This is because a `.ANY` region uses the maximum size when assigning sections. The maximum size might not be available at that point, because the size of all regions is not known until after the `.ANY` assignment.

The following example shows how to use `ImageLimit(`*region_name*`)` to place one execution region immediately after another:

Example 4-6 Placing an execution region after another

```
LR1 0x8000
{
    ER1 0x100000
    {
        *(+RO)
    }
}
LR2 0x100000
{
    ER2 (ImageLimit(ER1))               ; Place ER2 after ER1 has finished
    {
        *(+RW +ZI)
    }
}
```

4.23.1 Using +*offset* with expressions

A +*offset* value for an execution region is defined in terms of the previous region. You can use this as an input to other expressions such as `AlignExpr`. For example:

```
LR1 0x4000
{
    ER1 AlignExpr(+0, 0x8000)
    {
        ...
    }
}
```

By using `AlignExpr`, the result of `+0` is aligned to a `0x8000` boundary. This creates an execution region with a load address of `0x4000` but an execution address of `0x8000`.

4.23.2 See also

Concepts

- *Considerations when using a relative address +offset for load regions* on page 4-16
- *Considerations when using a relative address +offset for execution regions* on page 4-17
- *About Expression evaluation in scatter files* on page 4-30
- *Scatter files containing relative base address load regions and a ZI execution region* on page 4-36
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression usage in scatter files* on page 4-31
- *Expression rules in scatter files* on page 4-32
- *ScatterAssert function and load address related functions* on page 4-38
- *Symbol related function in a scatter file* on page 4-40
- *AlignExpr(expr, align) function* on page 4-42.

Using the Linker:

- *Image$$ execution region symbols* on page 7-6.

4.24 Scatter files containing relative base address load regions and a ZI execution region

You might want to place *Zero Initialized* (ZI) data in load region LR1, and use a relative base address for the next load region LR2, for example:

```
LR1 0x8000
{
    er_progbits +0
    {
        *(+RO,+RW) ; Takes space in the Load Region
    }
    er_zi +0
    {
        *(+ZI) ; Takes no space in the Load Region
    }
}
LR2 +0 ; Load Region follows immediately from LR1
{
    er_moreprogbits +0
    {
        file1.o(+RO) ; Takes space in the Load Region
    }
}
```

Because the linker does not adjust the base address of LR2 to account for ZI data, the execution region `er_zi` overlaps the execution region `er_moreprogbits`. This generates an error when linking.

To correct this, use the `ImageLimit()` function with the name of the ZI execution region to calculate the base address of LR2. For example:

```
LR1 0x8000
{
    er_progbits +0
    {
        *(+RO,+RW) ; Takes space in the Load Region
    }
    er_zi +0
    {
        *(+ZI) ; Takes no space in the Load Region
    }
}
LR2 ImageLimit(er_zi) ; Set the address of LR2 to limit of er_zi
{
    er_moreprogbits +0
    {
        file1.o(+RO) ; Takes space in the Load Region
    }
}
```

4.24.1 See also

Concepts

- *About Expression evaluation in scatter files* on page 4-30.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression usage in scatter files* on page 4-31

- *Expression rules in scatter files* on page 4-32
- *Execution address built-in functions for use in scatter files* on page 4-34.

Using the Linker:

- *Image$$ execution region symbols* on page 7-6.

4.25 ScatterAssert function and load address related functions

The `ScatterAssert(`*`expression`*`)` function can be used at the top level, or within a load region. It is evaluated after the link has completed and gives an error message if *`expression`* evaluates to false.

The load address related functions can only be used within the `ScatterAssert` function. They map to the three linker defined symbol values:

Table 4-3 Load address related functions

Function	Linker defined symbol value
`LoadBase(`*`region_name`*`)`	`Load$$`*`region_name`*`$$Base`
`LoadLength(`*`region_name`*`)`	`Load$$`*`region_name`*`$$Length`
`LoadLimit(`*`region_name`*`)`	`Load$$`*`region_name`*`$$Limit`

The parameter *`region_name`* can be either a load or an execution region name. Forward references are not permitted. The *`region_name`* can only refer to load or execution regions that have already been defined.

The following example shows how to use the `ScatterAssert` function to write more complex size checks than those permitted by the *`max_size`* of the region:

Example 4-7 Using ScatterAssert to check the size of multiple regions

```
LR1 0x8000
{
    ER0 +0
    {
        *(+RO)
    }
    ER1 +0
    {
        file1.o(+RW)
    }
    ER2 +0
    {
        file2.o(+RW)
    }
    ScatterAssert((LoadLength(ER1) + LoadLength(ER2)) < 0x1000)
                                        ; LoadLength is compressed size
    ScatterAssert((ImageLength(ER1) + ImageLength(ER2)) < 0x2000)
                                        ; ImageLength is uncompressed size
}
ScatterAssert(ImageLength(LR1) < 0x3000) ; Check uncompressed size of LoadRegion
```

4.25.1 See also

Concepts

- *About Expression evaluation in scatter files* on page 4-30
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Syntax of a scatter file* on page 4-4

- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression usage in scatter files* on page 4-31
- *Expression rules in scatter files* on page 4-32
- *Execution address built-in functions for use in scatter files* on page 4-34
- *Symbol related function in a scatter file* on page 4-40.

Using the Linker:

- *Load$$ execution region symbols* on page 7-7.

4.26 Symbol related function in a scatter file

The symbol related function, `defined(global_symbol_name)` returns zero if *`global_symbol_name`* is not defined and nonzero if it is defined.

Example 4-8 Conditionalizing a base address based on the presence of a symbol

```
LR1 0x8000
{
    ER1 (defined(version1) ? 0x8000 : 0x10000)   ; Base address is 0x8000
                                                 ; if version1 is defined
                                                 ; 0x10000 if not
    {
        *(+RO)
    }
    ER2 +0
    {
        *(+RW +ZI)
    }
}
```

4.26.1 See also

Concepts

- *About Expression evaluation in scatter files* on page 4-30
- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *Syntax of a scatter file* on page 4-4
- *Syntax of a load region description* on page 4-6
- *Syntax of an execution region description* on page 4-9
- *Expression usage in scatter files* on page 4-31
- *Expression rules in scatter files* on page 4-32
- *Execution address built-in functions for use in scatter files* on page 4-34
- *ScatterAssert function and load address related functions* on page 4-38.

4.27 Example of aligning a base address in execution space but still tightly packed in load space

This example uses a combination of pre-processor macros and expressions to copy tightly packed execution regions to execution addresses in a page-boundary. Using the `ALIGN` scatter-loading keyword aligns the load addresses of `ER2` and `ER3` as well as the execution addresses

Example 4-9 Aligning a base address in execution space but still tightly packed in load space

```
#! armcc -E
#DEFINE START_ADDRESS  0x100000
#DEFINE PAGE_ALIGNMENT 0x100000

LR1 0x8000
{
    ER0 +0
    {
        *(InRoot$$Sections)
    }
    ER1 START_ADDRESS
    {
        file1.o(*)
    }
    ER2 AlignExpr(ImageLimit(ER1), PAGE_ALIGNMENT)
    {
        file2.o(*)
    }
    ER3 AlignExpr(ImageLimit(ER2), PAGE_ALIGNMENT)
    {
        file3.o(*)
    }
}
```

4.27.1 See also

Concepts

- *About Expression evaluation in scatter files* on page 4-30.

Reference

- *Syntax of a load region description* on page 4-6
- *Load region attributes* on page 4-7
- *Syntax of an execution region description* on page 4-9
- *Execution region attributes* on page 4-11
- *AlignExpr(expr, align) function* on page 4-42
- *GetPageSize() function* on page 4-43
- *SizeOfHeaders() function* on page 4-44.

4.28 `AlignExpr(expr, align)` function

This function returns:

```
(expr + (align-1)) & ~(align-1))
```

where:

- `expr` is a valid address expression
- `align` is the alignment, and must be a positive power of 2.

It increases `expr` until it is:

`0` mod `align`

4.28.1 Example

This example aligns the address of `ER2` on an 8-byte boundary:

```
ER +0
{
    ...
}

ER2 AlignExpr(+0x8000,8)
{
    ...
}
```

4.28.2 Relationship with the `ALIGN` keyword

The following relationship exists between `ALIGN` and `AlignExpr`:

`ALIGN` keyword

Load and execution regions already have an `ALIGN` keyword:

- for load regions the `ALIGN` keyword aligns the base of the load region in load space and in the file to the specified alignment
- for execution regions the `ALIGN` keyword aligns the base of the execution region in execution and load space to the specified alignment.

`AlignExpr` Aligns the expression it operates on, but has no effect on the properties of the load or execution region.

4.28.3 See also

Reference

- *Execution region attributes* on page 4-11.

4.29 `GetPageSize()` function

Returns the page size. This is useful when used with `AlignExpr`

Returns the value of the internal page size that `armlink` uses in its alignment calculations. By default this value is set to `0x8000`, but you can change it with the `--pagesize` command-line option.

4.29.1 Example

This example aligns the base address of `ER` to a Page Boundary:

```
ER AlignExpr(+0, GetPageSize())
{
    ...
}
```

4.29.2 See also

Concepts

- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Reference

- *--pagesize=pagesize* on page 2-93
- *AlignExpr(expr, align) function* on page 4-42.

4.30 `SizeOfHeaders()` function

Returns the size of ELF Header plus the estimated size of the Program Header Table. This is useful when writing demand paged images to start code and data immediately after the ELF Header and Program Header Table.

4.30.1 Example

This example sets the base of `LR1` to start immediately after the ELF Header and Program Headers:

```
LR1 SizeOfHeaders(){    ...}
```

4.30.2 See also

Concepts

- *Example of aligning a base address in execution space but still tightly packed in load space* on page 4-41.

Using the Linker:

- *Demand paging* on page 4-23
- *About creating regions on page boundaries* on page 8-52.