

vlink portable multi-format linker

Frank Wille

February 2026

Table of Contents

1	General	1
1.1	Introduction	1
1.2	Legal	1
1.3	Contact	1
1.4	Installation	1
2	The Linker	3
2.1	Usage	3
2.2	Supported File Formats	3
2.3	Linker Options	9
2.3.1	Target specific options <code>amigahunk</code> , <code>amigaehf</code>	15
2.3.2	Target specific options <code>appleomf</code>	15
2.3.3	Target specific options <code>aoutmint</code> , <code>ataritos</code>	16
2.3.4	Target specific options <code>o65-02</code> , <code>o65-816</code>	16
2.3.5	Target specific options <code>oricmc</code>	17
2.3.6	Target specific options <code>os9-6809</code>	17
2.3.7	Target specific options <code>sinclairql</code>	17
2.3.8	Target specific options <code>rawbin</code>	18
2.3.9	Target specific options <code>vobj-be</code> , <code>vobj-le</code>	18
2.3.10	Target specific options <code>xfile</code>	18
3	Linker Scripts	19
3.1	Memory Layout	19
3.2	Output Sections	19
3.3	Program Headers	21
3.4	Commands	22
3.5	Functions	23
4	Appendix	25
4.1	Known Problems	25
4.2	Credits	25
4.3	Error Messages	25

1 General

1.1 Introduction

`vlink` is a portable linker which can be configured to support multiple input and output file formats. It even allows linking of input files with different formats within a single run and generates an output file format of your choice from it.

The linker supports linking with objects, object archives (static libraries) and shared objects or libraries. It can generate an executable file with or without additional information for dynamic linking, a shared object, or a new object file suitable for another linker pass.

Empty sections and other unused data are deleted to achieve a size-optimized output.

1.2 Legal

`vlink` is copyright 1995-2026 by Frank Wille.

This archive may be redistributed without modifications and used for non-commercial purposes.

An exception for commercial usage is granted, provided that the target OS is AmigaOS/68k. Resulting binaries may be distributed commercially without further licensing.

In all other cases you need my written consent.

1.3 Contact

Responsible for the current version of `vlink` and contact address in case of bug reports or support requests:

- Frank Wille (frank@phoenix.owl.de)

1.4 Installation

`vlink` comes as a stand-alone program, so no further installation is necessary. To use `vlink` with `vbcc`, copy the binary to `vbcc/bin`, following the installation instructions for `vbcc`.

2 The Linker

2.1 Usage

vlink links the object and library archive files given on the command line into a new object file. The output object file is either an executable program, a shared object suitable for loading at run-time, or an object file that can once again be processed by **vlink**.

Object files and library archives are processed in the order given on the command line. Unlike other linkers you will usually have to specify each library to link against only once, as **vlink** is smart enough to figure out all dependencies.

Internally, the linker distinguishes three section types:

code	Sections containing executable cpu instructions. Mapped as read-only on some systems.
data	Sections containing initialized data.
bss	Sections containing uninitialized data. Many systems will initialize them to zero on start. Usually only its size is stored in the output.

Additionally, a section has attributes like readable, writeable, executable, minimum alignment or target-specific attributes (small-data, etc.). Standard behaviour, in absence of a linker script, is to merge sections which have the same type and name. Attributes and linker options may prevent or force merging of certain sections, depending on the selected target format.

The file format of an input object file is determined automatically by the linker. The default output file format is compiled in (see **-v**) and may be changed by **-b**. Optionally, the default library search path can also be compiled in and is visible with **-v** as well.

The output file formats available can be configured at compile time. Refer to **config.h**. The default is to include all.

2.2 Supported File Formats

The following file formats are available:

amigahunk

The AmigaDOS hunk format for M68k. Requires AmigaOS 2.04 for option **-Rshort**. No shared objects. Small data offset 0x7ffe. Linker symbols:

- **_DATA_BAS_** (PhxAss)
- **_DATA_LEN_** (PhxAss)
- **_BSS_LEN_** (PhxAss)
- **_LinkerDB** (all)
- **__BSSBAS** (SAS/C, StormC)
- **__BSSLEN** (SAS/C, StormC)
- **___ctors** (SAS/C, StormC)
- **___dtors** (SAS/C, StormC)
- **_RESLEN** (SAS/C)

- `_RESBASE` (SAS/C)
- `_NEWDATA1` (SAS/C)
- `__DATA_BAS` (DICE-C)
- `__DATA_LEN` (DICE-C)
- `__BSS_LEN` (DICE-C)
- `__RESIDENT` (DICE-C)
- `___machtype` (GNU-gcc)
- `___text_size` (GNU-gcc)
- `___data_size` (GNU-gcc)
- `___bss_size` (GNU-gcc)

Automatic constructor/destructor function tables: `___ctors` and `___dtors` (will be mapped automatically to `__CTOR_LIST__` and `__DTOR_LIST__`). Fastcall-ABI constructors/destructors replace standard ABI ones, when using identical name and priority.

Referencing `_RESLEN` switches the hunk-format output module into Resident mode, which will append a special relocation table to the initialized part of the data-bss section and warns about absolute references from other sections. This mode can be used to create reentrant, "pure" programs, for use with the AmigaOS `resident` command.

Hunk-format executables can be linked with some restrictions. All sections will get an auto-generated, unique, name (specifying their type and their memory flags).

Supports `-Rstd` and `-Rshort`. The target-specific `-hunkattr` can be used to overwrite the memory flags of an input section. This format was called `amigaos` in former `vlink` versions.

- | | |
|-----------------------|---|
| <code>amigaehf</code> | An extension of the AmigaDOS hunk format for the PowerPC, 32-bit, big endian, as introduced by Haage&Partner GmbH for WarpOS. No executables (they are in <code>amigahunk</code> format) or shared objects. The same linker symbols, constructors/destructors as under <code>amigahunk</code> are supported. Additionally, <code>@_name</code> symbols will be created on demand (when referenced). Supports <code>-Rstd</code> , <code>-Rshort</code> and the target-specific <code>-hunkattr</code> . |
| <code>amsdos</code> | Absolute raw binary multi-file output (like <code>rawbin</code> option <code>-multifile</code>), but with a header for Amstrad/Schneider CPC computers. |
| <code>a.out</code> | Currently supported: <ul style="list-style-type: none"> – <code>aoutnull</code> (Default with standard relocs and undefined endianness) – <code>aoutbsd68k</code> (NetBSD/68k) – <code>aoutbsd68k4k</code> (NetBSD/68k 4K page size) – <code>aoutsun010</code> (SunOS 68010 and AmigaOS/Atari 68000/010) – <code>aoutsun020</code> (SunOS 68020 and AmigaOS/Atari 68020-68060) – <code>aoutbsdi386</code> (NetBSD/i386) – <code>aoutpc386</code> |

- `aoutmint` (Embeds `a.out` in TOS format for Atari MiNT executables, supports extra options like the `ataritos` format)
- `aoutjaguar` (M68k with special, word-swapped RISC relocations)

The minimal section alignment of `a.out` input and output files is determined by taking the destination format's minimal pointer alignment or the alignment defined by the `-minalign` option. Whatever is larger. Small data offset: `0x8000` (unused). Linker symbols: `__GLOBAL_OFFSET_TABLE_`, `__PROCEDURE_LINKAGE_TABLE_`, `__DYNAMIC`.

<code>applebin</code>	Absolute raw binary output, but with a 4-byte header prepended (containing load-address and length), as used by the <code>AppleCommander</code> to write the file onto an Apple DOS 3.3 disk image.
<code>appleomf</code>	Apple IIgs relocatable Object Module Format (OMF), as described in the File Format chapter of Apple IIGS Programmer's Workshop. Executables (load-files) only at the moment. The section's data model is recognized either by name (substring in section name: <code>".near"</code> or <code>".huge"</code> or <code>".zero"</code> or <code>".dpage"</code> , defaults to far) or by a character in the section's attributes: <code>'z'</code> for zero or direct-page, <code>'N'</code> for near, <code>'f'</code> for far and <code>'H'</code> for huge. Different data models will never be merged into one segment. Huge segments ignore bank size limits. Only Far segments pay attention to the <code>-maxsegsz</code> option. The linker symbol <code>__DBR_init</code> is provided to define the base address of the first near data segment, which can be used to initialize DBR. Target-specific options: <code>-maxsegsz=<size></code> , <code>-stack=<size></code> , <code>-version=<n></code> .
<code>ataricom</code>	Absolute raw binary output, but with a file header and section headers for Atari 8-bit computers (Atari 400, 600, 800, etc.).
<code>ataritos</code>	Atari-ST TOS executable file format and DRI object file format. Libraries are supported in native DRI and standard ar-format. Symbol tables may use either the HiSoft or the SozobonX name extension. Symbols in TOS executables may be section- or start-based (option <code>-tos-textbased</code> for compatibility with <code>MonST</code>). Additionally supports the target-specific options: <code>-tos-flags</code> , <code>-tos-fastload</code> , <code>-tos-fastram</code> , <code>-tos-fastalloc</code> , <code>-tos-private</code> , <code>-tos-global</code> , <code>-tos-super</code> , <code>-tos-readable</code> . The internal linker script defines <code>_LinkerDB</code> for small data and supports vbcc-style constructor/destructor tables in the data section (<code>__CTOR_LIST__</code> and <code>__DTOR_LIST__</code>).
<code>bbc</code>	Absolute raw binary output, but additionally writes an <code>.inf</code> -file for BBC Micro/Master emulators.
<code>bbc2</code>	Like <code>bbc</code> , but banked memory will be written to separate files (with a suitable linker script) and a loader script is generated.
<code>cbmprg</code>	Absolute raw binary output, but with a header for Commodore 8-bit computers (PET, VIC-20, 64, etc.).
<code>cbmreu</code>	Writes multiple image files for the REU memory expansion. Only the first one has a Commodore PRG header.

- cocoml** Absolute raw binary output, but with segment headers and a trailer for Tandy Color Computer machine language files.
- dragonbin** Absolute raw binary output, but with a header for Dragon DOS binary files, suitable for Dragon 32 and 64 computers.
- elf32amigaos** Identical to **elf32ppcbe**, but when doing dynamic linking it requires that also all references from shared objects are resolved at link time. This is due to a limitation of the AmigaOS4 dynamic link editor (**elf.library**).
- elf32arm** ELF (executable linkable format) for the ARM architecture. 32-bit, little endian. Small data offset: **0x1000**. Linker Symbols: **_SDA_BASE_**. Automatic constructor/destructor function tables will be placed into the sections **.ctors** and **.dtors**. Supports **-Rstd** and **-Radd**.
- elf32aros** ELF i386 32-bit little endian like **elf32i386**, but generates relocatable object files as executables. This format is used for the AROS (Amiga Research OS) operating system. Supports **-Rstd** and **-Radd**.
- elf32i386** ELF (executable linkable format) for Intel 386 and better, 32-bit, little endian. No small data. Automatic constructor/destructor function tables will be placed into the sections **.ctors** and **.dtors**. Supports **-Rstd** and **-Radd**.
- elf32jag** ELF (executable linkable format) for Atari Jaguar RISC, 32-bit, big endian. Small data offset: **0**. Linker symbols: **_SDA_BASE_**. Automatic constructor/destructor function tables will be placed into the sections **.ctors** and **.dtors**. Supports **-Rstd** and **-Radd**.
- elf32m68k** ELF (executable linkable format) for Motorola M68k, 32-bit, big endian. Small data offset: **0x8000**. Linker symbols: **_SDA_BASE_**. Automatic constructor/destructor function tables will be placed into the sections **.ctors** and **.dtors**. Supports **-Rstd** and **-Radd**.
- elf32morphos** Nearly identical to **elf32powerup**. Only difference is that **.sdata** and **.sbss** sections will not be merged as the MorphOS loader will take care of it. This format is used for MorphOS.
- elf32powerup** ELF PowerPC 32-bit big endian like **elf32ppcbe**, but generates relocatable object files as executables. This format is used for the PowerUp kernel. The linker symbol **_LinkerDB** is defined for **vbccppc**-compatibility. Small data offset: **0x8000**. This format was also called **elf32amiga** in former **vlink** versions.
- elf32ppcbe** ELF (executable linkable format) for PowerPC, 32-bit, big endian. Small data offset: **0x8000**. Linker symbols: **_SDA_BASE_** and **_SDA2_BASE** (EABI only).

Automatic constructor/destructor function tables will be placed into the sections `.ctors` and `.dtors`.

elf64x86 ELF (executable linkable format) for the x86_64 architecture. 64-bit, little endian. No small data. Automatic constructor/destructor function tables will be placed into the sections `.ctors` and `.dtors`. Supports `-Rstd` and `-Radd`.

foenixpgx-c02

PGX is simple, single-segment format for the 65C02/65816-based Foenix computers. The header defines the program's load address, which is also the start address. This format sets the cpu-type in the header to 65C02.

foenixpgx-816

PGX is simple, single-segment format for the 65C02/65816-based Foenix computers. The header defines the program's load address, which is also the start address. This format sets the cpu-type in the header to 65816.

foenixpgz-24

PGZ is a multi-segment format for the Foenix computers. The format is derived from binary format used by Western Design Center's C compiler. Every segment is stored with load address and size, and a start address may be defined. This format outputs the 'Z' type using 24-bit address and size specifications.

foenixpgz-32

PGZ is a multi-segment format for the Foenix computers. The format is derived from binary format used by Western Design Center's C compiler. Every segment is stored with load address and size, and a start address may be defined. This format outputs the 'z' type using 32-bit address and size specifications.

ihex Intel Hex format. No symbols. Output format only. Without a linker script, the raw binary will be relocated to base address 0.

jagsrv Absolute raw binary output, but with a header to make it load and execute via the Atari Jaguar SkunkBoard or the VirtualJaguar emulator.

o65-02 The o65 binary relocation format for the 6502 family V1.3, as defined by Andre Fachat. Supports reading and writing object files and executables, which may be relocatable. The following target-specific options are supported: `-o65-align`, `-o65-author`, `-o65-bsszero`, `-o65-cpu`, `-o65-fopts`, `-o65-name`, `-o65-paged`, `-o65-stack`.

o65-816 The o65 binary relocation format for 65816 processors V1.3, as defined by Andre Fachat. Supports reading and writing object files and executables, which may be relocatable. The following target-specific options are supported: `-o65-align`, `-o65-author`, `-o65-bsszero`, `-o65-fopts`, `-o65-name`, `-o65-paged`, `-o65-stack`.

oricmc Absolute raw binary output, but with a tape header for machine code files, suitable for ORIC-1, Atmos, Telestrat and Pravetz computers. The name written to the tape file is the output file name. Note, that a ".tap" extension will be removed from that name and its length will be limited to 15 characters. The following target-specific options are supported: `-autox`.

- os9-6809** OS-9 program modules for the 6809 processor, as defined by Microware Systems Corporation and the NitrOS-9 project. Code is always position-independent and reentrant. Relocation tables for data-text and data-data references are appended at the initialized data section in the module. These tables have to be processed by the program's startup code. vbcc-style constructor/destructor tables will be created and placed into the data section. By default, the module name will be the same as the output file name. No support for symbol tables. Supports target-specific options: `-os9-mem`, `-os9-name`, `-os9-ns`, `-os9-rev`.
- rawbin** Absolute raw binary file. The sections and base addresses have to be specified by a linker script (option `-T`). Gaps between sections are filled with 0-bytes (you may change that default with `-fill`), unless option `-coalesced` was given. With option `-multifile` each inter-section gap larger than 16 bytes leads to the creation of a new file with the section's name appended after a dot. Without a linker script, the raw binary will be relocated to base address 0. When option `-q` (keep relocs) has been specified, the linker will not execute absolute address relocations, but append a relocation offset table at the end of the file. The width of a word in this table matches the target's address size and uses the target's endianness. The first word defines the size of the following table in bytes. You may have to define a label in your code or linker script to access the end of your program for referencing this table. The default linker script defines the `__end` symbol. It follows a byte-stream for the relocation offsets. A byte between 1 and 255 represents the distance in bytes to the next relocation offset (starts at zero). A 0-byte indicates that the following word contains a distance greater than 255. Your startup code has to add the program's start address to the address values in all these locations. Warning: remaining relocations, like PC-relative, or other absolute relocations not matching the address size, will still be resolved by the linker! Which may be desired, or not. Supports target-specific options: `-coalesced`, `-fill`, `-multifile`.
- rawseg** Creates a raw binary file for each segment. Segments can be defined in a PHDR block of the linker script. It defaults to text and data segments. The segment names, their base address and length are written into the output file while the binary files get their segment name appended to the original file name. When option `-q` (keep relocs) has been specified, then additional files containing the relocation offsets are created. The first word in each file defines the number of relocations. The width of all words in this table matches the target's address size. Note, that only simple address relocations with the full address size are supported (no halfwords, etc.), which makes it nearly useless for certain targets, like the 6502 or RISC CPUs.
- sinclairql** Absolute raw binary output for the 68008-based Sinclair QL computer. Prepends a QDOS file header (default, `-qhdr` or an XTcc trailer (`-xtcc`)). These headers are mostly used directly by emulators or for file transfer. The following target-specific options are supported: `-qhdr`, `-xtcc`, `-stack=<n>`.

srec19	
srec28	
srec37	Motorola S-Record format. No symbols. Output format only. Without a linker script, the raw binary will be relocated to base address 0.
vobj-le	
vobj-be	VOBJ file format, generated by the <code>vasm</code> assembler. VOBJ is an object file format designed to support any little- or big-endian architecture with all their specific relocations.
wozmon	Simple text-based format that can be pasted as input for the monitor used e.g. in the Apple-I. No symbols. Output format only. The executable address will be appended at the end, so that the program can be started by a simple <code>R</code> command.
xfile	Human68k XFile format, as used on Sharp X68000 computers. Executables only at the moment. Symbol table supports absolute symbols and relocatable symbols from the text, data and bss segment. The format has no differentiation between local and global scope. The internal linker script defines <code>.LinkerDB</code> for small data and supports <code>vbcc</code> -style constructor/destructor tables in the data section (<code>__CTOR_LIST__</code> and <code>__DTOR_LIST__</code>). The following target-specific options are supported: <code>-x-high</code> .

2.3 Linker Options

Usually options and input file names can be mixed. Order of options may be important (e.g. when specifying a library with `-l` or a search path with `-L`).

The following general options are supported:

-Bdynamic	Specifies that linking against dynamic libraries can take place. If a library specifier like <code>-lx</code> appears on the command line, <code>vlink</code> searches for a library of the form <code>libx.so.n.m</code> (see the <code>-l</code> option) according to the search rules in effect. If such a file cannot be found a traditional archive is looked for. This option may appear anywhere on the command line and is complementary to <code>-Bstatic</code> .
-Bstatic	The counterpart of <code>-Bdynamic</code> . This option turns off dynamic linking for all library specifiers until a <code>-Bdynamic</code> is once again given. Any explicitly mentioned shared object encountered on the command line while this option is in effect is flagged as an error.
-Bshareable	Instructs the linker to build a shared object from the object files rather than a normal executable image.
-Bsymbolic	This option causes all symbolic references in the output to be resolved in this link-edit session. The only remaining runtime relocation requirements are base-relative relocations, ie. translation with respect to the load address. Failure to resolve any symbolic reference causes an error to be reported.

- Bforcearchive**
Force all members of library archives to be loaded, whether or not such members contribute a definition to any plain object files. Useful for making a shared library from an archive of PIC objects without having to unpack the archive.
- b targetname**
Specifies target file format for the output file. See also "Supported file formats".
- baseoff offset**
Defines section offset for base-relative relocations. The default offset is target-dependent (e.g. 0x7ffe for amigaos and 0x8000 for elf32m68k).
- C constructor-type**
Defines the type of constructor/destructor function names to scan for. Valid types are:
 - gnu GNU style constructors
 - vbcc vbcc style constructors: __INIT[_<pri>]_<name> / __EXIT..
 - vbccelf vbcc style constructors: __INIT[_<pri>]_<name> / _EXIT..
 - sasc SAS/C style constructors: __STI[_<pri>]_<name> / __STD..
- Crel**
Function references in the constructor/destructor tables (see above) are written as relative offsets to their current table position instead of absolute pointers. Useful for PC-relative code.
- clr-addunderscore**
No longer add a preceding underscore for the symbols of the following objects on the command line.
- clr-delunderscore**
No longer delete a preceding underscore for the symbols of the following objects on the command line.
- D linkersymbol[=value]**
Define the linker symbol `linkersymbol`, so it may be referenced by linker script expressions and from object code. The optional `value` is assigned to it, which defaults to 1.
- d**
- dc**
- dp**
Force allocation of common symbols, even when producing relocatable output (**-r** option).
- da**
Force allocation of address symbols (PowerOpen), even when producing relocatable output (**-r** option).
- di**
Force allocation of constructor and destructor tables, even when producing relocatable output (**-r** option).
- e entrypoint**
Defines the entry point of an executable and may be either a symbol or an absolute address. The linker will set the entry point by trying each of the following methods in order, stopping when the first succeeds:
 1. **-e** option

2. `ENTRY()` command in a linker script
 3. value of the symbol `_start`, if defined
 4. start of the first executable code section
 5. address 0
- EB** Presets big-endian mode for reading input and writing output.
- EL** Presets little-endian mode for reading input and writing output.
- export-dynamic**
Put all global symbols of the output file into the dynamic symbol table, making them visible for shared objects loaded on demand (e.g. by `dlopen()`).
- f flavour**
Adds a library-flavour. All flavours are cumulatively appended to each library search-path, whenever a library was specified with `-l`. Example: One search path and two flavours will search in:
1. `<lib-path>`,
 2. `<lib-path>/<flavour1>` and
 3. `<lib-path>/<flavour1>/<flavour2>`
- F filename**
A list of object file names is read from the specified file. Useful, if the number of objects exceeds the length of the command line.
- fixunnamed**
All unnamed sections will get a default name according to their section type (`.text`, `.data` and `.bss`).
- gc-all** Section garbage collection. Starting from the executable's entry point, determine all referenced sections and delete the unreferenced ones.
- gc-empty**
Delete all empty sections from an executable, which are not referenced from anywhere in the linked input files. Note: Before V0.15d `vlink` tried to do that always, but it didn't work in all cases.
- h** Prints a short description for all options. Includes the specific options for the target selected with `-b`. Example: `vlink -bo65-02 -h`
- interp interpreter-path**
Defines the name of the interpreter, which is usually the dynamic linker for dynamically linked ELF executables. Defaults to `/usr/lib/ld.so.1`.
- k** Keeps the original section order as found in the object files from the command line. Otherwise `vlink` links all code sections first, then all data and finally all bss, even when the first object starts with data. Has no meaning when using a linker script!
- L library-search-path**
Add path to the list of directories to search for libraries specified with the `-l` option. When a default search path was compiled in (see `-v`), then it is searched last, just before looking into the local directory.

-l library-specifier

This option specifies a library to be considered for inclusion in the output. If the **-Bdynamic** option is in effect, a shared library of the form **lib<spec>.so.m.n** (where **m** is the major, and **n** is the minor version number, respectively) is searched for first. The library with the highest version found in the search path is selected. If no shared library is found or the **-Bstatic** option is in effect, a library archive of the form **lib<spec>.a** is looked for in the library search path. For some formats, like **amigaos/amigaehf**, the libraries are called **<spec>.lib**. For all library specifiers on the command line a linker script is provided with **__VLINK_LIB_NAME** symbols, where **NAME** is the library specifier in upper case. The value is always 1.

-lineoffsets filename

Output source line offsets into a file in ASCII format. The output format first defines indexes for source file names, followed by all sections with a list of triples in the format: **src-index:line:offset**.

-M[file name]

Produce output about the mapping of sections from the input files and the values assigned to symbols in the output file. When the optional **file name** is missing the output goes to stdout.

-m

Enable special treatment of feature-mask suffixes in symbol names. A decimal number after the last **'.'** in a symbol name is stored as a feature-mask for symbol definitions. The masks in references to the symbol's name (sans suffix) are combined to a common requirement mask, which is used to find the best symbol to fulfill this requirement. Any reference to that symbol without a mask specification will disable that feature for all references. Also, when the requirement is not met by any masked symbol, then the normal symbol definition is taken.

-minalign alignment

Set a minimum alignment (number of bits which have to be zero) for all imported sections. The specified **alignment** value will only take effect when higher than the section's current alignment. It defaults to 0.

-mall

Merge all sections into a single output section. (Only when linking without a linker-script!)

-mattr

Merge all sections with the same attributes (readable, writeable, executable), memory model (small, large, etc.) and target-specific memory attributes. (Only when linking without a linker-script!)

-mrel

Automatically merge sections, when there are PC-relative references between them. (Only when linking without a linker-script!)

-mtype

Merge all sections of the same type (code, data, bss), even when their names or attributes differ. (Only when linking without a linker-script!)

-multibase

The default behaviour of **vlink** is to merge all sections which are referenced by base-relative addressing modes. This guarantees a single small data section, which can be accessed through a base register. If this is not desired - maybe

you have several base registers and small data sections - you can disable this behaviour by specifying `-multibase`.

- `-N oldname newname`
Rename all input sections named `oldname` into `newname`. This setting is valid for all the following input files and libraries on the command line and can be disabled with `-N oldname oldname`. Multiple `-N` options are allowed.
- `-n`
No page alignment of sections or segments in the final executable (`NMAGIC`).
- `-nostdlib`
Ignore default library search path, if one was compiled in.
- `-nowarn=n`
Do not display warning number `n`. May occur multiple times.
- `-o filename`
Specifies the name of the output file. Defaults to `a.out`.
- `-obe`
Use big-endian order on the octets (8-bit host bytes) in a target-byte, when writing output to the host's file system (default). Only valid for target architectures with more than 8 bits per byte.
- `-ole`
Use little-endian order on the octets (8-bit host bytes) in a target-byte, when writing output to the host's file system. Only valid for target architectures with more than 8 bits per byte.
- `-osec`
Output each section as an individual file. The file name given with `-o` will be ignored. Only available for some target formats: `rawbin`, `amsdos`, `cbmprg`.
- `-osec=basename`
Works like `-osec`, but each output file name will be preceded by `"basename."`.
- `-P symbol`
Protect a symbol from stripping. This doesn't work for all targets!
- `-q`
Emit relocations, even for absolute executables.
- `-R format`
Sets the relocation table format. Usually there is no need to change the default format defined by the target (`-b` option). Valid format strings are:
 - `std` standard format with addends in the code
 - `add` addends are stored in the relocation table
 - `short` relocation table with short offsets (e.g. 16 bit)
 Note that most targets only support one or two of these formats.
- `-r`
Produce relocatable object file, suitable for another linker pass.
- `-rpath library-search-path`
Add a directory to the runtime library search path. This is used when linking an ELF executable with shared objects. All `-rpath` arguments are concatenated and passed to the runtime linker, which uses them to locate shared objects at runtime.
- `-S`
Strip all debugger symbols from the output.

- s** Strip all symbols from the output.
- sc** Merge all code sections to a single code section (small code).
- sd** Merge all data and bss sections to a single data-bss section (small data).
- set-addunderscore**
Start adding a preceding underscore for the symbols of the following objects on the command line.
- set-delunderscore**
Start deleting a preceding underscore for the symbols of the following objects on the command line.
- shared** Instructs the linker to build a shared object from the object files rather than a normal executable image.
- soname name**
Sets the "real name" of a shared object or library. For ELF this will create the SONAME tag in the `.dynamic` section.
- symctrl=flags**
Control/filter the symbol file output by one or more flags added together. Currently the only valid flag is 4, to exclude local symbols.
- symfile filename**
Output all labels and absolute symbols, together with their value, into the given file. The output format of such a symbol/value line can be defined with **-symfmt**.
- symfmt formatstring**
Uses **formatstring** as a `printf`-style format string to output a symbol/value pair into the file given by **-symfile**. Any integer format character (`diouxX`) can be used for the value. The `s` character designates the position of the symbol's name. The default output format is `"0x%08llx:%s"`.
- T script** Specifies a linker script, which defines the mapping of input sections and their absolute locations in memory. The command language used is meant to be nearly identical to that used in GNU linker scripts, although not everything is implemented and there are a few additional commands. See Chapter 3 [Linker Scripts], page 19.
- Ttext addr**
Set the base address of the first section. It can be overridden by a linker script. Without a linker script it either sets the start address of the first section or of any section, depending on the output format.
- t** Trace the linker's file accesses.
- textbaserel**
Allow base-relative access on code sections. Otherwise the linker will display a warning.
- u symbol** Marks symbol as undefined in the first section which was found on the command line. This might trigger linking of additional modules from standard libraries. This is equivalent to the linker script command `EXTERN`.

- V version** Minimum major version of shared object to be linked behind this option.
- v** Prints **vlink** version string, default library search path and implemented target file formats.
- vicelabels filename** Generates a label address mapping file for the VICE debugger. This is a short-cut for a symbol file with **-symfmt "a1 C:%04lx .%s"**.
- w** Suppress all warning messages.
- wfail** The return code of **vlink** will no longer be 0 (success), when there was a warning. Errors always make the return code a failure.
- X** Discard local symbols in the input files that start with the letters 'L' or 'l', or with a dot.
- x** Discard all local symbols in the input files.
- y symbol** Trace the manipulations inflicted on symbol.
- Z** Do not move trailing zero-bytes in an initialized section into the uninitialized part of it, when generating executable output files. Usually the uninitialized part of a section is determined by the difference between the section's real size and file size (for those file formats which support it).

2.3.1 Target specific options **amigahunk**, **amigaehf**

- broken-debug** Try to ignore free-floating debug hunks, which are used in an illegal way like a section, together with relocations.
- hunkattr secname=value** Overwrite the memory attributes of all input sections named **secname** with **value**. For example allocate the **DATA** section in Chip-RAM: **-hunkattr DATA=2**. Extended memory attributes are supported.
- kick1** Make sure the executable file is compatible with Kickstart 1.x. It avoids 16-bit reloc offsets (ignores **-Rshort**), data-bss sections (same as **-Z**), 32-bit PC-relative relocs and warns about known bugs in the 1.x hunk-loader.

2.3.2 Target specific options **appleomf**

- maxsegs�ize=size** Define the maximum size in bytes for segments with multiple sections. A section which doesn't fit into the current segment is moved into the next.
- stack=size** Stack size in bytes to add to the DirectPage/Stack segment.
- version=n** Generate a version 1 (default) or version 2 OMF file.

2.3.3 Target specific options aoutmint, ataritos

- tos-flags value**
Set the 32 bit flags field of the Atari TOS header to **value**.
- tos-fastload**
Sets the fastload bit (0) in the TOS header.
- tos-fastram**
Sets the fastram bit (1) in the TOS header.
- tos-fastalloc**
Sets the fastalloc bit (2) in the TOS header.
- tos-private**
Sets the flags in the TOS header to mark memory space as private.
- tos-global**
Sets the flags in the TOS header to mark memory space as global (read/write by any process).
- tos-sozobonx**
Uses the Sozobon extension to write unlimited symbol names. Disables the Hisoft symbol name extension.
- tos-std dri**
Write a standard DRI symbol table, which limits symbol names to 8 characters. This is the default for object files. For TOS executables the HiSoft symbol name extension is enabled, which allows up to 22 characters.
- tos-super**
Sets the flags in the TOS header to mark memory space as read-writeable by processes in supervisor mode only.
- tos-readable**
Sets the flags in the TOS header to mark memory space as read-only for other processes.
- tos-textbased**
Writes text-based (offset to program start) DRI symbols to a TOS executable, like Devpac does. Otherwise symbol offsets are based on the section they are defined in.

2.3.4 Target specific options o65-02, o65-816

- o65-align val**
Set minimum alignment for all sections as number of least significant bits which have to be zero. **val** may be 0, 1, 2, 8. Default behaviour is to use the maximum alignment given by the input sections.
- o65-author name**
Store the given author name in the output.
- o65-bsszero**
Set a flag in the header which requests automatic clearing of the **.bss** section.

-o65-cpu cpmodel

Defines that the executable uses the instruction set of the given 6502-model `cpmodel`. Known models are: 6502, 65c02, 65sc02, 65ce02, nmos6502, 65816 (in 6502 emulation mode). Option is not available for `o65-816`, which is native 65816.

-o65-fopts

Enable informational header options, generated by the linker: file name, linker name and version, creation date.

-o65-name name

Set the "name" header option to `name`. Overwrites the real name which would be written by `-o65-fopts`.

-o65-paged

Make the output file use paged alignment and simplified paged relocations.

-o65-stack val

Store required stack size as `val` to the header.

2.3.5 Target specific options oricmc

-autox Set the auto-execute flag in the header, so the program starts automatically after loading.

2.3.6 Target specific options os9-6809**-os9-mem=val [K]**

Defines the size of the stack and the parameter area, which the linker will add to the permanent storage size in the module header. The value given in `val` is in bytes. You may specify the value in K-bytes by appending a 'k' character. The size of the stack and parameter area defaults to 1024 bytes for OS9/6809.

-os9-name=modname

Set the name of the OS-9 module to `modname`. Otherwise it defaults to the name of the output file or may be specified in the code, labeled by the symbol `__modname`.

-os9-ns Declare the OS-9 module as non-shareable and non-reentrant. It resets the shareable-flag in the module header, which is set otherwise.

-os9-rev=val

Set the revision in the OS-9 module header to `val`. Must be a value between 0 and 15. Defaults to zero.

2.3.7 Target specific options sinclairql

-qhdr Prepend a QDOS file header (default).

-xtcc Append an XTcc style trailer.

-stack=val

Set the stack size required by the program to `val`. Affects the dataspace size calculation. Defaults to 4096.

2.3.8 Target specific options rawbin

-coalesced

Memory gaps between sections are ignored. The following output section is written directly attached to the previous one.

-multifile

Memory gaps of more than 15 bytes between two output section create a new file. The file name has the format `outname.sectionname`. This option selects the former `rawbin2` target format.

-fill=val

Defines the fill-value (target-byte) for filling inter-section gaps. Defaults to zero.

2.3.9 Target specific options vobj-be, vobj-le

-vobj<N> Use vobj format version <N>. Currently supported are:

1. Base version.
2. Generally reduces file size.
3. Supports indirect symbols and absolute section addresses.

Note, that vlink will automatically increase the version to V3 when there are indirect symbols or section addresses in the output. Otherwise the default is V1.

-vobjcpu name

When writing a relocatable object file in VOBJ format (`-r` option), without reading any VOBJ input file, you may have to define the target CPU. Especially when target-specific relocations are used. Otherwise the default CPU is "generic". Example: `-vobjcpu PowerPC`

2.3.10 Target specific options xfile

-x-high Sets the high-address flag in the header.

3 Linker Scripts

3.1 Memory Layout

By default sections may be allocated in all available memory. You can define specific memory regions by using the `MEMORY` command. The syntax is:

```
MEMORY {
    memblockname (attr) : ORIGIN = org, LENGTH = len [,ID = id]
    ...
}
```

Defines one or more memory regions with start address `org` and a length of `len` bytes. The attributes in `(attr)` are optional and will be ignored by `vlink`, when specified (just for compatibility, at the moment). The keywords `ORIGIN` and `LENGTH` may be abbreviated down to a single character (`o=` and `l=`). Once a memory region is defined as `memblockname` the output of sections can be redirected into it by appending `>memblockname` at the end of a section definition. The memory ID (or bank ID) is optional and can be referenced by a special relocation type which is only supported in `VOBJ` format.

3.2 Output Sections

```
SECTIONS {
    ...
}
```

This is the only mandatory block in a linker script and is used to define the mapping of input sections to output sections, as well as their location in memory.

Within this block there may be symbol assignments, also for the location counter (`.`), commands and output section definitions.

A symbol assignment looks like

```
symbol = expression;
```

where `expression` may contain the usual arithmetic operations, other symbols and functions (See Section 3.5 [Functions], page 23). The special symbol `.` (dot) is the location counter and defines the current address (VMA) in memory where the following sections and data are placed.

An optional symbol assignment looks like

```
symbol =? expression;
```

where the difference to a normal symbol assignment is that `symbol` will only be defined with the given expression when it was not already defined before (in a line above, or on the command line with option `-D`).

All valid linker-script commands are described here: See Section 3.4 [Commands], page 22.

An output section definition has many optional attributes and looks like this in its complete form:

```
secname vma (BANKSIZE=sz) (NOLOAD) : AT(lma) {
    file/section-patterns and commands
    ...
}
```

```
} >region AT>lma-region :phdr =fill
```

Mandatory are only **secname**, the colon and the curly-braces. Everything else is optional.

secname Name of the output section to create at the address of the current location counter.

vma When given, defines the section's start address (VMA) as **vma** and also redefines the location counter.

(**BANKSIZE=***sz*[,*off*])

Optional. Sets the output section's bank size to **sz** bytes. Whenever an input section crosses bank-boundaries, align it to the beginning of the next bank. The optional **off** argument can be used to always start a bank **off** bytes behind the bank's usual starting address.

(**NOLOAD**) Optional. Avoids writing the section's contents into the output file. Usually this makes sense for uninitialized sections, like BSS.

AT(lma) Optionally sets the load-address (LMA) of the section to **lma**. Useful for initialized data loaded into ROM, which is copied to its real address in RAM during startup.

>region Optionally redirects this output section into memory region **region** (defined by the **MEMORY** command, See Section 3.1 [Memory], page 19). Each memory region has its own location counter!

AT>lma-region

Optionally load this output section into memory region **lma-region** (defined by the **MEMORY** command, See Section 3.1 [Memory], page 19). Each memory region has its own location counter!

:phdr Defines that the output section should go into program segment **phdr**. PHDR segments are used in ELF executables and in vlink's **rawseg** output target. Optional. Uses the last **phdr** when omitted.

=fill **fill** optionally defines a 16-bit pattern used to fill skipped or undefined regions in the output section (refer to **FILL16**).

Between the curly-braces there may be linker-script commands (See Section 3.4 [Commands], page 22), symbol assignments and one or multiple input section specifications. An input section specification consists of a single file-pattern (**fpat**) and one or multiple section-patterns, which looks like: **fpat(spat1 spat2...)**.

To match multiple files or sections the usual wildcards may be used for file- and section-patterns. The wildcard capabilities depend on the host operating system vlink is running on, so do not expect that anything more than '*' (match any string) and '?' (match any character) will work.

All matching input files and their matching input sections will be merged into the output section at this point, provided the output section has enough space (as defined by **MEMORY** region constraints).

The following functions are valid when specifying input section patterns:

KEEP(fpat(spat...))

Always keep these input sections in the output. Never delete them by any form of garbage collection (e.g. **-gc-all** or **-gc-empty**).


```

fpat(EXCLUDE_FILE(xpat...) spat...)
    Files matching to any of xpat will be excluded from the matching file pattern
    (fpat).

fpat(EXCLUDE_SECTION(xpat...) spat...)
    Section names matching to any of xpat will be excluded from the matching
    section patterns (spat).

fpat(SORT(spat...))
    Alias for SORT_BY_NAME.

fpat(SORT_BY_ALIGNMENT(spat...))
    Merge matching input sections sorted by alignment (descending).

fpat(SORT_BY_NAME(spat...))
    Merge matching input sections sorted by name (ascending).

fpat(SORT_BY_SIZE(spat...))
    Merge matching input sections sorted by size (descending).

fpat(REVERSE(SORT_BY_mode(spat...)))
    Reverse the following sorting order. Defaults to sort by name when the proper
    SORT command is missing.

```

Multiple **EXCLUDE** functions (of different kind) may be specified at once, but they need to be the first functions within the parentheses following the **fpat**.

It is also allowed to specify up to two sorting levels in a single pattern. For example **fpat(SORT_BY_NAME(SORT_BY_ALIGNMENT(spat...)))** would sort all sections by their name first, then sections with the same name by alignment.

3.3 Program Headers

```

PHDRS {
    phdrname type FILEHDR PHDRS AT(addr) FLAGS(flags);
    ...
}

```

Program headers are also known as segments and mainly used in ELF executables. Segments define a block of multiple sections with similar attributes (e.g. executable and read-only or read-write). The linker defines reasonable default Program Headers, but you may want to overwrite the default.

phdrname	Defines the segment's name, which may be used in an output section definition, using the :phdrname syntax.
type	The segment type may be PT_LOAD , PT_DYNAMIC , PT_INTERP , PT_NOTE , PT_SHLIB or PT_PHDR . Refer to the ELF-ABI documentation for a precise description. PT_LOAD defines a segment which is loaded into memory. PT_PHDR defines a segment which includes the program header itself.
FILEHDR	Set this optional attribute when the segment includes the (ELF) file header information.
PHDRS	Set this optional attribute when the segment includes the program header table. Typically set together with FILEHDR .

AT(addr) Optionally defines the segment's start address to be **addr**.

FLAGS(flags)

Optionally defines the segment permission as **flags**, where bit 0 means executable, bit 1 means writable and bit 2 means readable.

3.4 Commands

The following commands are currently supported in vlink linker scripts:

ASSERT(expression,"message")

Evaluate **expression** and print an assertion error, including the optional **message**, when zero.

BYTE(expression)

Insert a target byte at the current section address and assign the value of **expression** to it.

CONSTRUCTORS

Set the constructor/destructor function collection strategy to GNU-style constructors. They are usually already placed into **.ctors** and **.dtors** sections.

ENTRY(symbol)

symbol defines the entry point of program execution, which may be used by some executable file formats. It is also used to define the starting point for section garbage collection (**-gc-all** option).

EXTERN(symbol [symbol ...])

Define one or multiple symbols as undefined, which might trigger linking of additional modules from standard libraries. Refer to option **-u**.

FILL8(expression)

Specify an 8-bit fill-pattern, which is used to fill skipped regions in a section (e.g. by alignments or setting a new location counter).

FILL16(expression)

Specify a 16-bit fill-pattern, which is used to fill skipped regions in a section (e.g. by alignments or setting a new location counter). The **expression** is always written in big-endian order.

GROUP(file [file...])

For compatibility. Works just like **INPUT** in vlink.

INPUT(file [file...])

Define input files, which has exactly the same effect like on the command line. Specifying libraries needs a **-l** prefix. When there are also input files on the command line, the files specified here will be appended.

LONG(expression)

Insert four target-bytes at the current section address and assign the value of **expression**, using the target's endianness, to it.

OUTPUT_ARCH()

OUTPUT_FORMAT()

No meaning in vlink. Just for compatibility. Refer to option **-b** to define the output file format.

PROVIDE(symbol = expression)

The **symbol** will only be defined with **expression** when it is referenced from anywhere in the input files.

QUAD(expression)

Insert 8 target-bytes at the current section address and assign the value of **expression**, using the target's endianness, to it.

RESERVE(space)

Reserve **space** bytes at the current location counter, which are filled with the value given by **FILL8()** or **FILL16()** (defaults to zero).

SEARCH_DIR(path)

Appends **path** as an additional library search path. Has the same effect as **-L**.

SHORT(expression)

Insert two target-bytes at the current section address and assign the value of **expression**, using the target's endianness, to it.

SQUAD(expression)

Insert 8 target-bytes the current section address and assign the value of **expression**, using the target's endianness, to it.

VBCC_CONSTRUCTORS

Set the constructor/destructor function collection strategy to vbcc-style constructors (**__INIT[_<pri>]_<name> / __EXIT..**) and put them into the current section.

VBCC_CONSTRUCTORS_ELF

Set the constructor/destructor function collection strategy to vbcc-style ELF constructors (**_INIT[_<pri>]_<name> / _EXIT..**) and put them into the current section.

3.5 Functions

The following functions are currently supported in vlink linker scripts:

ADDR(sectionname)

Return the address (VMA) of the section named **sectionname**.

ALIGN(align)

Return the location counter (**.**), aligned to the next address which is a multiple of **align**.

LENGTH(memoryname)

Return the length of the memory region named **memoryname**.

LOADADDR(sectionname)

Return the loading-address (LMA) of the section named **sectionname**.

`MAX(exp1,exp2)`

Return the maximum value of the two expressions `exp1` and `exp2`.

`MIN(exp1,exp2)`

Return the minimum value of the two expressions `exp1` and `exp2`.

`ORIGIN(memoryname)`

Return the start address of the memory region named `memoryname`.

`SIZEOF(sectionname)`

Return the size of the section named `sectionname` in bytes.

`SIZEOF_HEADERS`

Return the size of the output file format's header in bytes.

4 Appendix

4.1 Known Problems

- Neither shared objects nor dynamically linked executables can be generated for `a.out` format.
- The following options are not really supported: `-S`, `-X`, `-Bsymbolic`
- Source level debugging support is missing for some formats.
- Many linker script commands are still missing.
- Default linker scripts are mostly missing, so you need to provide your own script by using the `-T` option.
- PHDR support for ELF is not perfect.

4.2 Credits

All those who wrote parts of the `vlink` distribution, made suggestions, answered my questions, tested `vlink`, reported errors or were otherwise involved in the development of `vlink` (in ascending alphabetical order, probably not complete):

- Karoly Balogh
- Volker Barthelmann
- Matthias Bock
- Dennis Boon
- Alexander Coers
- Romain Giot
- Stefan Haubenthal
- Mikael Kalms
- Chester Kollschen
- Miro Kropacek
- Jean-Paul Mari
- Gunther Nikl
- Thorsten Otto
- Keith S
- Jörg Strohmayr

4.3 Error Messages

1. Out of memory
2. Unrecognized option '%s'
3. Unknown link mode: %s
4. Unknown argument for option -d: %c
5. Option '-%c' requires an argument
6. No input files

7. File "%s" has a read error
8. Cannot open "%s"
9. Invalid target format "%s"
10. Directory "%s" could not be examined
11. %s: File format not recognized
12. "%s" is already an executable file
13. %s: File format corrupted
14. %s (%s): Illegal relocation type %d at %s+%x
15. %s: Unexpected end of section %s in %s
16. %s: %s appeared twice in %s
17. %s: Misplaced %s in %s
18. %s: Symbol definition %s in %s uses unsupported type %d
19. %s: Global symbol %s from %s is already defined in %s
20. %s: Unresolved reference to symbol %s in %s uses unsupported type %d
21. %s (%s+0x%x): Reference to undefined symbol %s
22. Attributes of section %s were changed from %s in %s to %s in %s
23. %s: %s expected
24. %s (%s+0x%x): Illegal relative reference to %s+0x%llx
25. %s (%s+0x%x): %dbit %s reference to %s+0x%llx (value to write: 0x%llx) out of range
26. %s (%s+0x%x): Referenced absolute symbol %s=0x%llx + 0x%llx (value to write: 0x%llx) doesn't fit into %d bits
27. %s (%s+0x%x): Illegal relative reference to symbol %s
28. %s (%s+0x%x): Relative reference to relocatable symbol %s=0x%llx + 0x%llx (value to write: 0x%llx) doesn't fit into %d bits
29. Can't create output file %s
30. Target file format doesn't support shared objects
31. Error while writing to %s
32. Target %s: Unsupported relocation type %s (offset=%d, size=%d, mask=%llx) at %s+0x%x
33. Target %s: Can't reproduce symbol %s, which is a %s%s%s
34. Option '%s' requires an argument
35. %s (%s+0x%x): Calculated value 0x%llx doesn't fit into relocation type %s (offset=%d, size=%d, mask=0x%llx)
36. UNUSED
37. %s: Malformatted archive member %s
38. %s: Empty archive ignored
39. %s: %s doesn't support shared objects in library archives
40. %s: %s doesn't support executables in library archives
41. %s (%s): Illegal format / file corrupted

42. %s: Consistency check for archive member %s failed
43. %s: Invalid ELF section header index (%d) in %s
44. %s: ELF section header #%d has illegal offset in %s
45. %s: ELF section header string table has illegal type in %s
46. %s: ELF section header string table has illegal offset in %s
47. %s: ELF program header table in %s was ignored
48. %s: ELF section header type %d in %s is not needed in relocatable objects
49. %s: Illegal section offset for %s in %s
50. %s: ELF %s table has illegal type in %s
51. %s: ELF %s table has illegal offset in %s
52. %s: %s in %s defines relocations relative to a non-existing section with index=%d
53. %s: Symbol %s, defined in %s, has an invalid reference to a non-existing section with index=%d
54. %s: Illegal symbol type %d for %s in %s
55. %s: Symbol %s has illegal binding type %d in %s
56. %s: Symbol %s in %s is multiply defined
57. %s: Merging a code section with name "__MERGED"
58. Relative references between %s section "%s" and %s section "%s" (%s) force a combination of the two
59. Can't define %s as ctors/dtors label. Symbol already exists.
60. %s: ELF section header type %d in %s is not needed in shared objects
61. %s: Endianness differs from previous objects
62. Target file format doesn't support relocatable objects
63. Predefined limits of destination memory region %s for output section %s were exceeded (0x%llx)
64. Section %s(%s) was not recognized by target linker script
65. %s line %d: Unknown keyword <%s> ignored
66. %s line %d: '%c' expected
67. %s line %d: Absolute number expected
68. %s line %d: Keyword <%s> expected
69. %s line %d: GNU command <%s> ignored
70. %s line %d: Unknown memory region <%s>
71. %s line %d: Multiple constructor types in output file
72. %s line %d: Unknown keyword <%s>
73. %s line %d: Assertion failed: %s
74. %s line %d: SECTIONS block defined twice
75. %s line %d: Segment %s is closed and can't be reused
76. %s line %d: Address overrides specified %cMA memory region
77. %s line %d: Segment %s must include both, FILEHDR and PHDR

- 78. %s line %d: Missing argument
- 79. %s line %d: Undefined section: <%s>
- 80. %s line %d: Section %s was assigned to more than one PT_LOAD segment
- 81. Multiple use of section <%s> in linker script
- 82. Intermediate uninitialized sections in ELF segment <%s> (first=<%s>, last=<%s>) will be turned into initialized
- 83. Section <%s> (0x%llx-0x%llx) conflicts with ELF segment <%s> (currently: 0x%llx-0x%llx)
- 84. %s: QMAGIC is deprecated and will no longer be supported
- 85. %s: a.out %s table has illegal offset or size in %s
- 86. %s: a.out %s table size in <%s> is not a multiple of %d
- 87. %s: a.out symbol name has illegal offset %ld in %s
- 88. %s: a.out symbol %s has illegal binding type %d in %s
- 89. %s: a.out relocations without an appropriate section in %s
- 90. %s: illegal a.out relocation in section %s of %s at offset 0x%08lx: <pcrel=%d len=%d ext=%d brel=%d jmptab=%d rel=%d copy=%d>
- 91. %s: illegal a.out external reference to symbol %s in %s, which is no external symbol
- 92. %s: illegal nlist type %lu in a.out relocation in section %s of %s at offset 0x%08lx
- 93. Target %s: Common symbol %s is unreferenced and will disappear
- 94. Target file format doesn't support executable files
- 95. %s: a.out relocation <pcrel=%d len=%d ext=%d brel=%d jmptab=%d rel=%d copy=%d> is treated as a normal relocation in section %s of %s at offset 0x%08lx
- 96. %s: size %d for a.out symbol %s in %s was ignored
- 97. Target %s: %s section must not be absent for a valid executable file
- 98. Target %s: Section %s is overlapping %s
- 99. %s line %d: Illegal PHDR type: <%s>
- 100. %s line %d: <%s> behind SECTIONS ignored
- 101. %s line %d: Address symbol '.' invalid outside SECTIONS block
- 102. %s line %d: Reference to non-absolute symbol <%s> outside SECTIONS
- 103. %s line %d: Division by zero
- 104. %s line %d: Unknown symbol or function: <%s>
- 105. %s line %d: No function-calls allowed here
- 106. %s line %d: Symbol <%s> is not yet assigned
- 107. %s line %d: Command <%s> not allowed outside SECTIONS block
- 108. %s line %d: Address symbol '.' cannot be provided
- 109. %s line %d: Symbol <%s> already defined
- 110. %s line %d: Only absolute expressions may be assigned outside SECTIONS block
- 111. %s line %d: Unknown PHDR: <%s>
- 112. %s (%s+0x%x): Cannot resolve reference to %s, because section %s was not recognized by the linker script

- 113. %s (%s): %d bits per byte are not supported
- 114. %s (%s): %d bytes per target-address are not supported
- 115. %s (%s): Relocation type %d (offset=%lld, bit-offset=%d bit-size=%d mask=0x%llx referring to symbol <%s> (type %d) is not supported
- 116. %s (%s): Symbol type %d for <%s> in section %s is not supported
- 117. %s (%s+0x%x): Cannot resolve %s reference to %s, because host section %s is invalid
- 118. %s: Malformatted ELF %s section in %s
- 119. %s: Ignoring junk at end of ELF %s section in %s
- 120. %s (%s+0x%x): Relocation based on missing %s section
- 121. %s (%s+0x%x): Base-relative reference to code section
- 122. Relocation table format not supported by selected output format - reverting to %s's standard
- 123. Unknown relocation table format '%s' ignored
- 124. Target %s: multiple small-data sections not allowed
- 125. .ctors/.dtors spread over multiple sections
- 126. Dynamic symbol reference not supported by target %s
- 127. %s: ELF symbol name has illegal offset 0x%lx in %s
- 128. %s: Unknown endianness defaults to %s-endian. Consider using -EB/-EL
- 129. Resetting the same attribute for section %s
- 130. Bad assignment after option '%s'
- 131. Need a valid symbolic entry when using -gc-all
- 132. Executable code section in first object required when using -gc-all
- 133. Unsupported absolute relocation (offs=%lld pos=%d siz=%d msk=0x%llx) in resident data section
- 134. %s (%s+0x%x): Absolute reference to resident data section (%s)
- 135. %s line %d: Undefined memory region: <%s>
- 136. Executable section <%s> in data segment not allowed
- 137. Not enough space for the module header (%u of %u)
- 138. Target %s: multiple %s sections not allowed: <%s> and <%s>
- 139. %s: symbol index %u is out of range
- 140. %s: %s is chained
- 141. Maximum file option size exceeded (%u)
- 142. %s: Ignoring weak symbol %s
- 143. %s: Unexpected relocations for section with index=%d
- 144. Bad error number: %d
- 145. Error number %d is not a warning
- 146. %s (%s): alternating bits per byte in object files (from %d to %d)
- 147. %s (%s): alternating bytes per address in object files (from %d to %d)
- 148. Endianness is unknown. Default to host endianness.

- 149. Mismatching target address sizes in input/output formats
- 150. %s: Hunk format corrupted: DEBUG hunk used like a section with name "%s" in unit "%s". Trying to ignore
- 151. %s: Duplicate con/destructor name %s definition ignored
- 152. Warnings treated as errors
- 153. %s: Kickstart 1.x cannot initialize bss sections >256k to zero
- 154. No CPU defined for VOBJ output. Consider using -vobjcpu
- 155. Indirect symbol %s -> %s not allowed in %s
- 156. Alternating CPU definitions from VOBJ input - keeping the first one (%s)
- 157. Bad symbol file format
- 158. No space to fit section %s(%s) with end address %#llx into memory region
- 159. %s (%s+%#lx): Reference from overlaid section %s to %s (symbol %s) is not allowed (NOCROSSREFS)
- 160. Target %s: Too many symbols for selected output format
- 161. Invalid version. Assuming V%d.%d
- 162. Target %s: Output section %s exceeds maximum size %lu
- 163. %s: No valid bank size defined
- 164. %s (%s): Section exceeds maximum bank size of %lu
- 165. %s (%s+%#lx): output section %s (referenced symbol: %s) has no memory id defined, assuming zero
- 166. %s (%s+%#lx): reference to memory id of relocatable symbol %s=%#llx%c%#llx (value to write: %s%#llx) doesn't fit into %d bits
- 167. %s line %d: Maximum of %d sorting levels exceeded
- 168. %s line %d: Multiple EXCLUDE_%s commands in one pattern
- 169. %s: Indirect target symbol for %s does not exist