A guide to the internals of the GNU linker

Per Bothner, Steve Chamberlain, Ian Lance Taylor, DJ Delorie
Cygnus Support

Cygnus Support
2.10.91
TeXinfo 1999-09-25.10

This file documents the internals of the GNU linker `ld`. It is a collection of miscellaneous information with little form at this point. Mostly, it is a repository into which you can put information about GNU `ld` as you discover it (or as you design changes to `ld`).

1 The 'README' File

Check the '`README`' file; it often has useful information that does not appear anywhere else in the directory.

2 How linker emulations are generated

Each linker target has an *emulation*. The emulation includes the default linker script, and certain emulations also modify certain types of linker behaviour.

Emulations are created during the build process by the shell script '`genscripts.sh`'.

The '`genscripts.sh`' script starts by reading a file in the '`emulparams`' directory. This is a shell script which sets various shell variables used by '`genscripts.sh`' and the other shell scripts it invokes.

The '`genscripts.sh`' script will invoke a shell script in the '`scripttempl`' directory in order to create default linker scripts written in the linker command language. The '`scripttempl`' script will be invoked 5 (or, in some cases, 6) times, with different assignments to shell variables, to create different default scripts. The choice of script is made based on the command line options.

After creating the scripts, '`genscripts.sh`' will invoke yet another shell script, this time in the '`emultempl`' directory. That shell script will create the emulation source file, which contains C code. This C code permits the linker emulation to override various linker behaviours. Most targets use the generic emulation code, which is in '`emultempl/generic.em`'.

To summarize, '`genscripts.sh`' reads three shell scripts: an emulation parameters script in the '`emulparams`' directory, a linker script generation script in the '`scripttempl`' directory, and an emulation source file generation script in the '`emultempl`' directory.

For example, the Sun 4 linker sets up variables in '`emulparams/sun4.sh`', creates linker scripts using '`scripttempl/aout.sc`', and creates the emulation code using '`emultempl/sunos.em`'.

Note that the linker can support several emulations simultaneously, depending upon how it is configured. An emulation can be selected with the `-m` option. The `-V` option will list all supported emulations.

2.1 'emulparams' scripts

Each target selects a particular file in the 'emulparams' directory by setting the shell variable targ_emul in 'configure.tgt'. This shell variable is used by the 'configure' script to control building an emulation source file.

Certain conventions are enforced. Suppose the targ_emul variable is set to *emul* in 'configure.tgt'. The name of the emulation shell script will be 'emulparams/*emul*.sh'. The 'Makefile' must have a target named 'e*emul*.c'; this target must depend upon 'emulparams/*emul*.sh', as well as the appropriate scripts in the 'scripttempl' and 'emultempl' directories. The 'Makefile' target must invoke GENSCRIPTS with two arguments: *emul*, and the value of the make variable tdir_*emul*. The value of the latter variable will be set by the 'configure' script, and is used to set the default target directory to search.

By convention, the 'emulparams/*emul*.sh' shell script should only set shell variables. It may set shell variables which are to be interpreted by the 'scripttempl' and the 'emultempl' scripts. Certain shell variables are interpreted directly by the 'genscripts.sh' script.

Here is a list of shell variables interpreted by 'genscripts.sh', as well as some conventional shell variables interpreted by the 'scripttempl' and 'emultempl' scripts.

SCRIPT_NAME
: This is the name of the 'scripttempl' script to use. If SCRIPT_NAME is set to *script*, 'genscripts.sh' will use the script 'scripteml/*script*.sc'.

TEMPLATE_NAME
: This is the name of the 'emultemlp' script to use. If TEMPLATE_NAME is set to *template*, 'genscripts.sh' will use the script 'emultempl/*template*.em'. If this variable is not set, the default value is 'generic'.

GENERATE_SHLIB_SCRIPT
: If this is set to a nonempty string, 'genscripts.sh' will invoke the 'scripttempl' script an extra time to create a shared library script. Section 2.2 [linker scripts], page 3.

OUTPUT_FORMAT
: This is normally set to indicate the BFD output format use (e.g., '"a.out-sunos-big"'. The 'scripttempl' script will normally use it in an OUTPUT_FORMAT expression in the linker script.

ARCH
: This is normally set to indicate the architecture to use (e.g., 'sparc'). The 'scripttempl' script will normally use it in an OUTPUT_ARCH expression in the linker script.

ENTRY
: Some 'scripttempl' scripts use this to set the entry address, in an ENTRY expression in the linker script.

TEXT_START_ADDR
: Some 'scripttempl' scripts use this to set the start address of the '.text' section.

`NONPAGED_TEXT_START_ADDR`
: If this is defined, the '`genscripts.sh`' script sets `TEXT_START_ADDR` to its value before running the '`scripttempl`' script for the `-n` and `-N` options (see Section 2.2 [linker scripts], page 3).

`SEGMENT_SIZE`
: The '`genscripts.sh`' script uses this to set the default value of `DATA_ALIGNMENT` when running the '`scripttempl`' script.

`TARGET_PAGE_SIZE`
: If `SEGMENT_SIZE` is not defined, the '`genscripts.sh`' script uses this to define it.

`ALIGNMENT`
: Some '`scripttempl`' scripts set this to a number to pass to `ALIGN` to set the required alignment for the `end` symbol.

2.2 '`scripttempl`' scripts

Each linker target uses a '`scripttempl`' script to generate the default linker scripts. The name of the '`scripttempl`' script is set by the `SCRIPT_NAME` variable in the '`emulparams`' script. If `SCRIPT_NAME` is set to *script*, `genscripts.sh` will invoke '`scripttempl/`*script*`.sc`'.

The '`genscripts.sh`' script will invoke the '`scripttempl`' script 5 to 8 times. Each time it will set the shell variable `LD_FLAG` to a different value. When the linker is run, the options used will direct it to select a particular script. (Script selection is controlled by the `get_script` emulation entry point; this describes the conventional behaviour).

The '`scripttempl`' script should just write a linker script, written in the linker command language, to standard output. If the emulation name–the name of the '`emulparams`' file without the '`.sc`' extension–is *emul*, then the output will be directed to '`ldscripts/`*emul*`.`*extension*' in the build directory, where *extension* changes each time the '`scripttempl`' script is invoked.

Here is the list of values assigned to `LD_FLAG`.

`(empty)`
: The script generated is used by default (when none of the following cases apply). The output has an extension of '`.x`'.

`n`
: The script generated is used when the linker is invoked with the `-n` option. The output has an extension of '`.xn`'.

`N`
: The script generated is used when the linker is invoked with the `-N` option. The output has an extension of '`.xbn`'.

`r`
: The script generated is used when the linker is invoked with the `-r` option. The output has an extension of '`.xr`'.

`u`
: The script generated is used when the linker is invoked with the `-Ur` option. The output has an extension of '`.xu`'.

`shared`
: The '`scripttempl`' script is only invoked with `LD_FLAG` set to this value if `GENERATE_SHLIB_SCRIPT` is defined in the '`emulparams`' file. The '`emultempl`' script must arrange to use this script at the appropriate time, normally when

the linker is invoked with the `-shared` option. The output has an extension of '`.xs`'.

`c` The '`scripttempl`' script is only invoked with `LD_FLAG` set to this value if `GENERATE_COMBRELOC_SCRIPT` is defined in the '`emulparams`' file or if `SCRIPT_NAME` is `elf`. The '`emultempl`' script must arrange to use this script at the appropriate time, normally when the linker is invoked with the `-z combreloc` option. The output has an extension of '`.xc`'.

`cshared` The '`scripttempl`' script is only invoked with `LD_FLAG` set to this value if `GENERATE_COMBRELOC_SCRIPT` is defined in the '`emulparams`' file or if `SCRIPT_NAME` is `elf` and `GENERATE_SHLIB_SCRIPT` is defined in the '`emulparms`' file. The '`emultempl`' script must arrange to use this script at the appropriate time, normally when the linker is invoked with the `-shared -z combreloc` option. The output has an extension of '`.xsc`'.

Besides the shell variables set by the '`emulparams`' script, and the `LD_FLAG` variable, the '`genscripts.sh`' script will set certain variables for each run of the '`scripttempl`' script.

`RELOCATING`
: This will be set to a non-empty string when the linker is doing a final relocation (e.g., all scripts other than `-r` and `-Ur`).

`CONSTRUCTING`
: This will be set to a non-empty string when the linker is building global constructor and destructor tables (e.g., all scripts other than `-r`).

`DATA_ALIGNMENT`
: This will be set to an `ALIGN` expression when the output should be page aligned, or to '`.`' when generating the `-N` script.

`CREATE_SHLIB`
: This will be set to a non-empty string when generating a `-shared` script.

`COMBRELOC`
: This will be set to a non-empty string when generating `-z combreloc` scripts to a temporary file name which can be used during script generation.

The conventional way to write a '`scripttempl`' script is to first set a few shell variables, and then write out a linker script using `cat` with a here document. The linker script will use variable substitutions, based on the above variables and those set in the '`emulparams`' script, to control its behaviour.

When there are parts of the '`scripttempl`' script which should only be run when doing a final relocation, they should be enclosed within a variable substitution based on `RELOCATING`. For example, on many targets special symbols such as `_end` should be defined when doing a final link. Naturally, those symbols should not be defined when doing a relocateable link using `-r`. The '`scripttempl`' script could use a construct like this to define those symbols:

```
${RELOCATING+ _end = .;}
```

This will do the symbol assignment only if the `RELOCATING` variable is defined.

The basic job of the linker script is to put the sections in the correct order, and at the correct memory addresses. For some targets, the linker script may have to do some other operations.

For example, on most MIPS platforms, the linker is responsible for defining the special symbol `_gp`, used to initialize the `$gp` register. It must be set to the start of the small data section plus `0x8000`. Naturally, it should only be defined when doing a final relocation. This will typically be done like this:

```
${RELOCATING+ _gp = ALIGN(16) + 0x8000;}
```

This line would appear just before the sections which compose the small data section (‘`.sdata`’, ‘`.sbss`’). All those sections would be contiguous in memory.

Many COFF systems build constructor tables in the linker script. The compiler will arrange to output the address of each global constructor in a ‘`.ctor`’ section, and the address of each global destructor in a ‘`.dtor`’ section (this is done by defining `ASM_OUTPUT_CONSTRUCTOR` and `ASM_OUTPUT_DESTRUCTOR` in the `gcc` configuration files). The `gcc` runtime support routines expect the constructor table to be named `__CTOR_LIST__`. They expect it to be a list of words, with the first word being the count of the number of entries. There should be a trailing zero word. (Actually, the count may be -1 if the trailing word is present, and the trailing word may be omitted if the count is correct, but, as the `gcc` behaviour has changed slightly over the years, it is safest to provide both). Here is a typical way that might be handled in a ‘`scripttempl`’ file.

```
${CONSTRUCTING+ __CTOR_LIST__ = .;}
${CONSTRUCTING+ LONG((__CTOR_END__ - __CTOR_LIST__) / 4 - 2)}
${CONSTRUCTING+ *(.ctors)}
${CONSTRUCTING+ LONG(0)}
${CONSTRUCTING+ __CTOR_END__ = .;}
${CONSTRUCTING+ __DTOR_LIST__ = .;}
${CONSTRUCTING+ LONG((__DTOR_END__ - __DTOR_LIST__) / 4 - 2)}
${CONSTRUCTING+ *(.dtors)}
${CONSTRUCTING+ LONG(0)}
${CONSTRUCTING+ __DTOR_END__ = .;}
```

The use of `CONSTRUCTING` ensures that these linker script commands will only appear when the linker is supposed to be building the constructor and destructor tables. This example is written for a target which uses 4 byte pointers.

Embedded systems often need to set a stack address. This is normally best done by using the `PROVIDE` construct with a default stack address. This permits the user to easily override the stack address using the `--defsym` option. Here is an example:

```
${RELOCATING+ PROVIDE (__stack = 0x80000000);}
```

The value of the symbol `__stack` would then be used in the startup code to initialize the stack pointer.

2.3 ‘emultempl’ scripts

Each linker target uses an ‘`emultempl`’ script to generate the emulation code. The name of the ‘`emultempl`’ script is set by the `TEMPLATE_NAME` variable in the ‘`emulparams`’ script. If the `TEMPLATE_NAME` variable is not set, the default is ‘`generic`’. If the value of `TEMPLATE_NAME` is *template*, ‘`genscripts.sh`’ will use ‘`emultempl/`*template*`.em`’.

Most targets use the generic ‘`emultempl`’ script, ‘`emultempl/generic.em`’. A different ‘`emultempl`’ script is only needed if the linker must support unusual actions, such as linking against shared libraries.

The 'emultempl' script is normally written as a simple invocation of cat with a here document. The document will use a few variable substitutions. Typically each function names uses a substitution involving EMULATION_NAME, for ease of debugging when the linker supports multiple emulations.

Every function and variable in the emitted file should be static. The only globally visible object must be named ld_*EMULATION_NAME*_emulation, where *EMULATION_NAME* is the name of the emulation set in 'configure.tgt' (this is also the name of the 'emulparams' file without the '.sh' extension). The 'genscripts.sh' script will set the shell variable EMULATION_NAME before invoking the 'emultempl' script.

The ld_*EMULATION_NAME*_emulation variable must be a struct ld_emulation_xfer_struct, as defined in 'ldemul.h'. It defines a set of function pointers which are invoked by the linker, as well as strings for the emulation name (normally set from the shell variable EMULATION_NAME and the default BFD target name (normally set from the shell variable OUTPUT_FORMAT which is normally set by the 'emulparams' file).

The 'genscripts.sh' script will set the shell variable COMPILE_IN when it invokes the 'emultempl' script for the default emulation. In this case, the 'emultempl' script should include the linker scripts directly, and return them from the get_scripts entry point. When the emulation is not the default, the get_scripts entry point should just return a file name. See 'emultempl/generic.em' for an example of how this is done.

At some point, the linker emulation entry points should be documented.

3 A Walkthrough of a Typical Emulation

This chapter is to help people who are new to the way emulations interact with the linker, or who are suddenly thrust into the position of having to work with existing emulations. It will discuss the files you need to be aware of. It will tell you when the given "hooks" in the emulation will be called. It will, hopefully, give you enough information about when and how things happen that you'll be able to get by. As always, the source is the definitive reference to this.

The starting point for the linker is in 'ldmain.c' where main is defined. The bulk of the code that's emulation specific will initially be in emultempl/*emulation*.em but will end up in e*emulation*.c when the build is done. Most of the work to select and interface with emulations is in ldemul.h and ldemul.c. Specifically, ldemul.h defines the ld_emulation_xfer_struct structure your emulation exports.

Your emulation file exports a symbol ld_*EMULATION_NAME*_emulation. If your emulation is selected (it usually is, since usually there's only one), ldemul.c sets the variable *ld_emulation* to point to it. ldemul.c also defines a number of API functions that interface to your emulation, like ldemul_after_parse which simply calls your ld_*EMULATION*_emulation.after_parse function. For the rest of this section, the functions will be mentioned, but you should assume the indirect reference to your emulation also.

We will also skip or gloss over parts of the link process that don't relate to emulations, like setting up internationalization.

After initialization, main selects an emulation by pre-scanning the command line arguments. It calls ldemul_choose_target to choose a target. If you set choose_target to ldemul_default_target, it picks your target_name by default.

main calls ldemul_before_parse, then parse_args. parse_args calls ldemul_parse_args for each arg, which must update the getopt globals if it recognizes the argument. If the emulation doesn't recognize it, then parse_args checks to see if it recognizes it.

Now that the emulation has had access to all its command-line options, main calls ldemul_set_symbols. This can be used for any initialization that may be affected by options. It is also supposed to set up any variables needed by the emulation script.

main now calls ldemul_get_script to get the emulation script to use (based on arguments, no doubt, see Chapter 2 [Emulations], page 1) and runs it. While parsing, ldgram.y may call ldemul_hll or ldemul_syslib to handle the HLL or SYSLIB commands. It may call ldemul_unrecognized_file if you asked the linker to link a file it doesn't recognize. It will call ldemul_recognized_file for each file it does recognize, in case the emulation wants to handle some files specially. All the while, it's loading the files (possibly calling ldemul_open_dynamic_archive) and symbols and stuff. After it's done reading the script, main calls ldemul_after_parse. Use the after-parse hook to set up anything that depends on stuff the script might have set up, like the entry point.

main next calls lang_process in ldlang.c. This appears to be the main core of the linking itself, as far as emulation hooks are concerned(*). It first opens the output file's BFD, calling ldemul_set_output_arch, and calls ldemul_create_output_section_statements in case you need to use other means to find or create object files (i.e. shared libraries found on a path, or fake stub objects). Despite the name, nobody creates output sections here.

(*) In most cases, the BFD library does the bulk of the actual linking, handling symbol tables, symbol resolution, relocations, and building the final output file. See the BFD reference for all the details. Your emulation is usually concerned more with managing things at the file and section level, like "put this here, add this section", etc.

Next, the objects to be linked are opened and BFDs created for them, and ldemul_after_open is called. At this point, you have all the objects and symbols loaded, but none of the data has been placed yet.

Next comes the Big Linking Thingy (except for the parts BFD does). All input sections are mapped to output sections according to the script. If a section doesn't get mapped by default, ldemul_place_orphan will get called to figure out where it goes. Next it figures out the offsets for each section, calling ldemul_before_allocation before and ldemul_after_allocation after deciding where each input section ends up in the output sections.

The last part of lang_process is to figure out all the symbols' values. After assigning final values to the symbols, ldemul_finish is called, and after that, any undefined symbols are turned into fatal errors.

OK, back to main, which calls ldwrite in 'ldwrite.c'. ldwrite calls BFD's final_link, which does all the relocation fixups and writes the output bfd to disk, and we're done.

In summary,

- main() in 'ldmain.c'
- 'emultempl/*EMULATION*.em' has your code
- ldemul_choose_target (defaults to your target_name)
- ldemul_before_parse
- Parse argv, calls ldemul_parse_args for each
- ldemul_set_symbols

- ldemul_get_script
- parse script
  - may call ldemul_hll or ldemul_syslib
  - may call ldemul_open_dynamic_archive
- ldemul_after_parse
- lang_process() in 'ldlang.c'
  - create output_bfd
  - ldemul_set_output_arch
  - ldemul_create_output_section_statements
  - read objects, create input bfds - all symbols exist, but have no values
  - may call ldemul_unrecognized_file
  - will call ldemul_recognized_file
  - ldemul_after_open
  - map input sections to output sections
  - may call ldemul_place_orphan for remaining sections
  - ldemul_before_allocation
  - gives input sections offsets into output sections, places output sections
  - ldemul_after_allocation - section addresses valid
  - assigns values to symbols
  - ldemul_finish - symbol values valid
- output bfd is written to disk

4 Some Architecture Specific Notes

This is the place for notes on the behavior of ld on specific platforms. Currently, only Intel x86 is documented (and of that, only the auto-import behavior for DLLs).

4.1 Intel x86

ld can create DLLs that operate with various runtimes available on a common x86 operating system. These runtimes include native (using the mingw "platform"), cygwin, and pw.

auto-import from DLLs

1. With this feature on, DLL clients can import variables from DLL without any concern from their side (for example, without any source code modifications). Auto-import can be enabled using the --enable-auto-import flag, or disabled via the --disable-auto-import flag. Auto-import is disabled by default.
2. This is done completely in bounds of the PE specification (to be fair, there's a minor violation of the spec at one point, but in practice auto-import works on all known variants of that common x86 operating system) So, the resulting DLL can be used with any other PE compiler/linker.

3. Auto-import is fully compatible with standard import method, in which variables are decorated using attribute modifiers. Libraries of either type may be mixed together.
4. Overhead (space): 8 bytes per imported symbol, plus 20 for each reference to it; Overhead (load time): negligible; Overhead (virtual/physical memory): should be less than effect of DLL relocation.

Motivation

The obvious and only way to get rid of dllimport insanity is to make client access variable directly in the DLL, bypassing the extra dereference imposed by ordinary DLL runtime linking. I.e., whenever client contains someting like

```
mov dll_var,%eax,
```

address of dll_var in the command should be relocated to point into loaded DLL. The aim is to make OS loader do so, and than make ld help with that. Import section of PE made following way: there's a vector of structures each describing imports from particular DLL. Each such structure points to two other parellel vectors: one holding imported names, and one which will hold address of corresponding imported name. So, the solution is de-vectorize these structures, making import locations be sparse and pointing directly into code.

Implementation

For each reference of data symbol to be imported from DLL (to set of which belong symbols with name <sym>, if __imp_<sym> is found in implib), the import fixup entry is generated. That entry is of type IMAGE_IMPORT_DESCRIPTOR and stored in .idata$3 subsection. Each fixup entry contains pointer to symbol's address within .text section (marked with __fuN_<sym> symbol, where N is integer), pointer to DLL name (so, DLL name is referenced by multiple entries), and pointer to symbol name thunk. Symbol name thunk is singleton vector (__nm_th_<symbol>) pointing to IMAGE_IMPORT_BY_NAME structure (__nm_<symbol>) directly containing imported name. Here comes that "om the edge" problem mentioned above: PE specification rambles that name vector (OriginalFirstThunk) should run in parallel with addresses vector (FirstThunk), i.e. that they should have same number of elements and terminated with zero. We violate this, since FirstThunk points directly into machine code. But in practice, OS loader implemented the sane way: it goes thru OriginalFirstThunk and puts addresses to FirstThunk, not something else. It once again should be noted that dll and symbol name structures are reused across fixup entries and should be there anyway to support standard import stuff, so sustained overhead is 20 bytes per reference. Other question is whether having several IMAGE_IMPORT_DESCRIPTORS for the same DLL is possible. Answer is yes, it is done even by native compiler/linker (libth32's functions are in fact resident in windows9x kernel32.dll, so if you use it, you have two IMAGE_IMPORT_DESCRIPTORS for kernel32.dll). Yet other question is whether referencing the same PE structures several times is valid. The answer is why not, prohibiting that (detecting violation) would require more work on behalf of loader than not doing it.

5 GNU Free Documentation License

GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the

Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
```

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Table of Contents