Programmer's Guide to `cereal`

Miloslav Trmač

mitr@volny.cz

Programmer's Guide to `cereal`
by Miloslav Trmač

Table of Contents

Chapter 1. Introduction

The main strength of `cereal`, its ability to emulate connected sets of devices, can rarely be fully exploited without writing custom modules emulating devices specific to your application. This document aims to provide enough information to get you started writing these modules and associated KDE UI. The last chapter describes interfaces you'll need for writing a new front-end to the emulation engine.

This document assumes reasonable user's knowledge of `cereal` (do read the complete User's Tutorial), knowledge of the C programming language in its latest standardized version known as C99, and ability to use libxml2 for saving and loading data (which takes about fifteen minutes to learn). For writing KDE UI extensions, ability to program KDE programs is obviously essential.

This document is only meant to be an overview, assuming you can find the actual interface definitions in the header files. You are also encouraged to use provided modules as examples (remember, **grep** is your friend).

Additionally, in the `sample` directory you can find two modules which were used in first real usage of `cereal`. While they can be viewed as an example how *not* to do things (namely combining UI and emulation code), they are included also to show how easy it can be.

Chapter 2. General Conventions

Errors are reported by returning a value described in the function synopsis. The responsibility to inform the user lies with the first function encountering the error condition, therefore if you call a function and receive the error code, you can just pass it to your caller without telling the user again. Errors are reported by calling the `error ()` function, which provided by the front-end.

> **Note:** The KDE front-ends (**cereal_kde** and **cereal_khwconf**) sometimes (quite often actually) need to try whether a particular operation is available or not without displaying any error message, or displaying it in less obtrusive way (i.e. in the Evaluate/Modify dialog box). This is achieved by indirecting through `error_handler`, which can be locally overridden to redirect the error messages.

For memory allocation in C use the provided `xmalloc ()`, `xrealloc ()`, `xstrdup ()` and `xxmlMalloc ()`, unless you are prepared to handle out-of-memory conditions gracefully. The provided functions check the result and abort when the allocation fails.

For reading unsigned integers from strings, you may want to use the `get_num ()` function which is simpler to use than `strtoul ()` and unlike `strtoul ()` also checks that there are no trailing characters left.

Chapter 3. `cereal` Modules

To implement a `cereal` module, you need to create a cereal_module structure which describes the module. To create a built-it module, you need to add your module to `cereal` linking process and add it to module_list.h. This will cause the module to be automatically registered on `cereal` startup. To create a dynamically-loaded one, create a shared library with a function `struct cereal_module *register_self (void)`, which calls `cereal_module_register ()` for your cereal_module and returns a pointer to it. This library needs to be named `libcereal`*`foo`*`.so` (where *`foo`* is any string) and placed in a directory given by a `CEREAL_MODULE_DIR` environment variable.

The cereal_module structure contains a few function pointers and description of ports the module provides. You can view the cereal_module as an object with virtual methods—except it is done in C. The following methods have to be defined:

Name	Description
mi_new	Called by the back-end to create a new instance of your module. This instance is by convention represented in a structure called mi (for Module Instance) and is completely private to your implementation. After creating the instance, return a pointer to it. In the unlikely event you don't need to keep any state, allocate a dummy non-null pointer using malloc (1) (note that malloc (0) may return `NULL`).
mi_delete	Called by the back-end to destroy an instance created by a previous call to mi_new. Perform needed cleanups and free the data.
set_option	Called to set an option of the given instance. Options can be changed in the **cereal_khwconf** module properties dialog. The set of options is your choice. If you need to pass complicated commands to your module which can not be reasonably implemented using ports, you can define options that are hidden from the user and used by your front-end to communicate with the emulated module. This is used in the 8051 module to allow the front-end to load an Intel HEX file to the internal program memory.
get_option	The obvious counterpart of set_option.

Name	Description
save_setup	Called to save the current setup to an XML file. The *setup* means properties of your object that don't change during emulation. This usually means only the options. Don't save the port connections here, this is done automatically by the back-end. Saving and loading data in XML is simple (and boring) work, see `8051/8051.c` for a largish example. You should select an XML namespace for your module and save all data not defined in the DTDs in `doc/dtd` using this namespace.
load_setup	The counterpart of save_setup. If your save_setup function just saves the options in a format <ns:option>value</ns:option>, you can use the `generic_module_load_setup_options ()` function instead of parsing the XML on your own. Other helpers to note include `get_xml_num ()` and `get_xml_enum ()`.
save_state	Called to save current emulation state. This does not include the setup, the back-ends saves both in a single file by calling both save_setup and save_state. For saving scheduling events (see below), use `schedule_save_event ()`.
load_state	The counterpart of load_state. For loading scheduling events use `schedule_load_state ()`.

It may make sense for set_option, get_option, load_setup and save_setup to do nothing. In that case, you can initialize the cereal_module members using generic_module_… functions (you can't just leave the pointers `NULL`).

You should already know that modules communicate with each other using ports and ports are grouped in spaces. Ports in a space should be logically grouped, and they all must have the same *width*,the number of bits that are transfered in one operation. The cereal_module structure contains a module_width structure for each allowed port width (which is internally represented by enum port_width, which currently allows 1, 8 and 16 bits). Thus any given port is completely identified by the module, width, space (zero-based index) and port (zero-based index).

> **Note:** It hasn't been explicitly stated yet, so here we go: `cereal` can only represent digital information with a simple, definite value. It can't directly represent analog computers or undefined states when the logical value is changing.

In the module_width structure, you need to fill in a pointer to array of structures describing individual port spaces with given width, and number of entries in this array. These entries are of type struct space and contain just pointers to functions. First of them is a get_size function, which returns number of ports in this space. Usually, this will be a constant, but it may also depend on a module option (such as the data_mem_size option of the 8051 module). The size of the space may not change during emulation (it must depend only on the module setup).

The space structure contains also two sets of function pointers, one per each port type (read, write, display, modify). The get_fn[] pointers are used to get functions used to access your ports (i.e. get_fn[PORT_TYPE_READ] is used to get a function that returns the current value of the port, to be displayed to the user). The set_fn[] pointers are used to connect other modules: if you connect for writing port A/A/A to port B/B/B (in that order), the get_fn[PORT_TYPE_WRITE] functions is called for the port B/B/B and the result is passed to the set_fn[PORT_TYPE_WRITE] function for port A/A/A. As a result, whenever port A/A/A has a value to write, it will call the function, which will handle the write on behalf of port B/B/B.

The functions are represented as struct port_fn, which contain the needed function pointer together with the destination module instance pointer and a function-specific data. There are two reasons a pure function pointer is not enough: First, you'll usually want to use a common function for the whole address space (i. e. when the address space represents bytes stored in a memory chip), so you need to preserve the port number given to the get_fn[] function. To understand why the instance pointer is needed, recall what the function of sfr_ext space in the 8051 module is. The get_fn[] of 8051 sfr space is implemented by returning the function connected (via set_fn[]) to the corresponding port of the sfr_ext space. Thus the following accesses (which really transfer data) go directly to the module connected on sfr_ext port without passing through the 8051 sfr—sfr_ext combination on each access.

The get_fn[] and set_fn[] pointers are in some ways different from the other function pointers used in `cereal`. First, the pointers can be `NULL`, which means that the operation is not supported. For a port to make sense, you need to support at least one type of access, though (unless the port serves as a placeholder to keep a relation between port addresses and some externally given addressing scheme (8051 SFR addresses)). Another difference is that the set_fn[] functions just return error code without telling the user.

In your module, you need to implement the "action" functions returned by get_fn[] and store port_fn structures for the ports that you need to connect to. In the set[] fn functions you then return either 0 if OK, `ENOTSUP` if the particular port doesn't support this type of access, or `EBUSY` if the port is already connected.

Now that you know how to create a module and how to make it communicate, it is time to make it do some work. Although some modules (the simple bit_report and bit_constant) modules can work with just immediately handling read/write/modify/display requests, your module will quite often need to simulate a separate process running in time (e.g. the uart module sending data). This is done by representing the process you emulate as a state machine, which can react to port accesses and timer expirations.

The timer expirations are represented by creating *scheduling events*. Schedule event is a timer that can be armed to trigger at a specified time. Each time the timer triggers, a handler function associated with the event is called. You can allocate scheduling events in your mi_new function by calling `schedule_new ()` and delete it in your mi_delete function by calling `schedule_delete ()`. To "arm" the event, call `schedule_add ()`, specifying the handling function and relative time after which the event triggers. At any time you can cancel the armed event by calling `schedule_cancel ()`. Usually, you'll want to arm one or more events in mi_new function to start your process (in fact, `cereal` needs at least one such module to do anything at all), but you may also want to arm an event in response to writing to one of your ports.

The back-end keeps the "emulation time" in the `cereal_time` variable, which you can copy any time to get a "snapshot". At a later time, you can call `schedule_difftime ()` to measure time between the snapshot and current time (this can be useful for example for measuring frequency on one of your input ports).

Sometimes, you'll want to report an error or a warning. Use the above-mentioned `error ()`, and also mark the fact by setting the bit `ER_ERROR` or `ER_WARNING` in the variable `emulation_result`. The front-ends should stop emulating when an error occurs, and they look at the `emulation_result` to do so. For similar reasons, if you are emulating a CPU and you have finished emulating an instruction (as defined by the usual meaning of the "Step" command), set the `ER_INSN` bit.

After creating the module, you need to make the vast possibilities available to the user. Create a *`module_name`*`.xml` in the `xml` directory. See the other files in that directory and `doc/dtd/cereal_module.dtd` for detailed information.

Chapter 4. KDE UI extensions

KDE UI extensions are implemented as dynamically loaded KParts plugins to KCMainWin, the emulator main window. Using KParts mostly amounts to copying the boilerplate code. You create a class derived from KLibFactory which can instantiate your plugin. This plugin checks that it is really connected to KCMainWin and then plugs its actions to its interface using the XMLGUI mechanism.

> **Note:** When copying the boilerplate code, don't use LDFLAGS=$(KDE_PLUGIN), because the plugin references symbols in the main executable. Instead, use the flags defined by $(KDE_PLUGIN) *without the -no-undefined flag*.

Usually, you'll want your module to provide an advanced interface to instances of a particular module type. To do this, check whether the module type is available at all by calling `cereal_module_find ()`, which returns a pointer to its cereal_module structure. Then, when the user invokes this interface, you can call `KCMainWin::selectModule ()` to let the user select the particular module instance that should be used. For a trivial example, see `KC8051::loadProgram ()`.

If you want to create a window, you should create it as a child of the widget returned by `KCMainWindow::viewParent ()`. This will automatically insert the window in the MDI framework. Your window should also in most cases inherit KCWindow and implement the abstract functions. This will cause your window to get notifications whenever the emulated state changes (if your window causes change of emulation state, you have to report this by calling `KCMainWin::updateViews ()`), and to be closed when the user loads a different file.

It is nice to the user to save the state of your interface extension to the XML file the emulation state is saved to, so that the user doesn't have to reenter breakpoints each time he runs `cereal`. To do this, implement the KCUIStateHandler interface (it is probably simplest to do so in your KParts plugin), its `saveState ()` and `loadState ()` methods, and register the handler by calling `KCMainWin::registerStateHandler ()`. As a convenience, your windows inherited from KCWindow have also a `saveState` method. Thus if all state you need to save is associated to your KCWindow descendants, you should implement `KCWindow::saveState ()` to save state created with that particular window, leave `KCUIStateHandler::saveState ()` empty and in `KCWindow::loadState ()` look for all window states saved, recreate the windows and restore their state.

You may find useful these `cereal`-specific widgets: KCLineEdit and KCListView are variants of KLineEdit and KListView with additional signals, KCExprLineEdit is an KLineEdit with a **Port...** button which allows the user to easily add port references. KCPortEdit is a KLineEdit with a label above which works as an Evaluate/Modify dialog box, except that the expression is fixed and invisible to the user. Finally KCBitEdit is a widget similar to KCPortEdit, except that it is used to modify a particular bit of the expression.

For the "real" work, you can use the interfaces described in the next chapter. When there is no special interface, just invoke the needed functionality directly (i.e. set_option and get_option module methods). The core of `cereal` is not written in C++ and does not have trivial wrapper functions around every data member.

Chapter 5. `cereal` Front-end Interface

`cereal` does not impose a specific structure to your front-end. For most purposes, you can view the `cereal` core as a library which you can use when you need to.

Module Handling

To create modules, you need to find the module type you want to operate on. This can be done either by searching by for it by name using `cereal_module_find ()`, or browsing `cereal_module_list`, the list of available module types.

Then you can create a module instance. Internally, module instances are usually passed around using struct me_entry, which contains everything needed to handle a module—the module type, name and instance data. Thus, you can create a module using `me_new`, delete it using `me_delete ()`, or rename it using `me_rename ()`. Module names are (mostly for simplicity) restricted to C-like identifiers. If you want to check whether a given name is allowed (for example in a validator of an edit control), use `me_name_ok ()`.

Once a module is created, you can obtain a pointer to its me_entry using `me_find ()` (which doesn't report a not-found error to the user) or `me_get ()` (which does). To handle options of the module, call the set_option and get_option functions directly. All module instances can be enumerated by browsing the `me_list`.

Port Space Handling

Port spaces are mostly a grouping of ports by a common set of get_fn[] and set_fn[] functions, so there is no need to care about them for the sole purpose of emulating. However, spaces are better presented to the user using human-readable names instead of a tuple (width, index). The names are defined in the module XML file and functions `space_create_name ()` and `space_find_name ()` allow you to directly transform space names and indexes.

Note that these functions (and whole `cereal` core) use for representing port widths enum port_width, not the integral values. Therefore, use `WIDTH_8` instead of just `8`. To communicate with the user, you can convert widths between these formats using `port_width_value` array and `get_port_width ()` function.

Port Handling

Ports themselves are more interesting than port spaces. A port of a given width can be identified by its module instance and its space and port indexes, which are grouped in struct port_id.

Similarly to spaces, ports can be also named in the module XML file. Use `port_create_name ()` and `port_find_name ()` to convert names and indexes. On a slightly higher level, to convert a port name in the `cereal` expression format of ***module/space/port***, use `port_find`.

Once you have a determined a port, you can read its value (using the "display" function, without side-effects) using `port_disp_W ()` and set it (using the "modify" function) using `port_set_W ()`, where *W* is 1, 8 or 16, depending on the port width. Usually it is easier to use `cereal` expressions interface described later.

When communicating with the user, you'll want to present human-readable names for the port types, which are internally represented by the `PORT_TYPE_*` values. To do this, use the `port_type_name` array and `port_type_find ()` function.

Port Connection Handling

A port connection is represented by a struct connect_entry, which contains the connection type, width and the two ports. All current connections are in `connect_list`. To connect two ports, fill in a “template” and pass it to `port_connect ()`, to disconnect them, use `port_disconnect ()`.

When creating a GUI for connection modification, it might be undesirable that modifying a connection (by disconnecting it and creating a new one) may change address of its connect_entry. In that case, disconnect the connection using `port_do_disconnect ()`, modify its members and reconnect it using `port_do_connect ()`. Also when creating the GUI, you’ll want to allow the users to cut the time spent creating the connections in half by allowing them to create, modify and delete the connections in read/write pairs. When you have one connection and want to find the other in its read/write pair (if it exists), call `port_peer_connection ()`.

Expression Handling

To evaluate a `cereal` expression, use `expr_eval ()`, to modify one expression to a value of another (as in the Evaluate/Modify dialog), use `expr_modify`. If you are often evaluating the same expression over and over, you should compile it by calling `expr_compile ()` which compiles the expression to a P-code, and then just pass the P-code to `expr_exec ()`. Evaluating compiled expression was about five times faster than parsing the expression each time for the expressions I have tested.

Breakpoint Handling

Breakpoint is represented by a struct breakpoint, which contains the expression both in text and compiled form and a place for front-end-specific data, (such as a breakpoint number in a `gdb`-like interface). Create a breakpoint using `breakpoint_new ()`, delete it using `breakpoint_delete ()`, change it in-place using `breakpoint_change ()`.

To evaluate a breakpoint, you can use `expr_eval ()` directly, or use `breakpoint_eval ()`, which will return just 0 or 1 (error in evaluation is taken as 0). All breakpoints are in `breakpoint_list`, to check whether some of them triggered use `breakpoint_check ()` (which also handles the “triggers only when was zero before” semantics, unlike `breakpoint_eval ()`).

Initialization, Finalization, Saving and Loading

The first thing you do with the `cereal` core should be calling `setup_init ()` (which registers all available module types) and the last thing you do should be calling `setup_destroy ()`, freeing used memory.

To save and load the "setup" (as in **cereal_khwconf**), use `setup_load ()` and `setup_save ()`. To save and load current emulation state (together with the used setup), use `state_load ()` and `state_save ()`. The two latter functions also allow you to save front-end-specific state along with the emulation state. Note however, that this front-end-specific state should not be needed for emulation, because it can be lost when the emulation state is opened in another front-end.

Starting the Emulation

Emulation is done by handling armed scheduling events in chronological order. To do so, call `schedule_run ()`, which will handle the first event in the queue. If there are no events, it returns `ER_HALT`. You may want to check for this condition in advance using `schedule_pending ()`.

In case you wonder, yes, this section has been placed almost at the absolute end intentionally. I wanted you to at least skim through all the interfaces, you'll need most of them anyway.

8051-specific Interfaces

If it is customary in your CPU architecture to store programs in the Intel hex format, use `load_hex_file ()`, `load_hex_stdio ()`, `load_xml_stdio ()` and `save_hex_xml ()` functions.

The 8051 disassembler used in **cereal_disasm** and the 8051 KDE UI extension is available as `i8051_disasm ()`.

Chapter 6. Happy Hacking!

Appendix A. GNU Free Documentation License

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 dispbibute 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[1] 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/[2].

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

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 © 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[3], to permit their use in free software.

Notes

1. http://www.gnu.org/fsf/fsf.html

2. http://www.gnu.org/copyleft
3. http://www.gnu.org/copyleft/gpl.html