Using libRaptorQ library: RAW interface

August 4, 2018

Abstract

libRaptorQ is a C++11 implementation of the RaptorQ Forward Error Correction. It includes two interfaces, one derived from the RFC6330 standard, and one RAW API for a more flexible and easy usage.

This document is only about the RAW interface.

The implementation was started as a university laboratory project, and will be later used and included in Fenrir, the maintainer's master thesis project.

Contents

1 Contacts **3**

2 Build & install **3**

2.1 Get the source code 3

2.2 Dependencies 4

2.3 Build & Install 4

3 Working with RaptorQ **7**

3.1 Theory (you really need this) 7

3.2 RaptorQ: Blocks & Symbols 7

3.3 C++ interface 8

3.3.1 Caching precomputations 9

3.3.2 The Encoder 10

3.3.3 Symbols 12

3.3.4 The Decoder 13

4 GNU Free Documentation License **16**

1 Contacts

The main development and dicussions on the project, along with bug reporting, happens on the main website.

Mailing Lists — Mailing lists are available at https://fenrirproject.org/lists
The two mailing lists are for development and announcements, but due to the low traffic of the development mailing list, it can also be used by users for questions on the project.

IRC — Since there are not many developers for now, the main irc channel is #**fenrirproject** on freenode

2 Build & install

2.1 Get the source code

This document follows the 1.0 stable release.

You can get the tarballs from the main website here:
https://fenrirproject.org/Luker/libRaptorQ/tags

Or you can check out the repository (warning: development in progress):

```
$ git clone --recurse-submodules \
        https://fenrirproject.org/Luker/libRaptorQ.git
```

You can also get it from github:

```
$ git clone --recurse-submodules \
        https://github.com/LucaFulchir/libRaptorQ.git
```

GPG verification: Once you have cloned it, it's always a good thing to check the repository gpg signatures, so you can import my key with:

```
$ gpg --keyserver pgp.mit.edu --recv-key \
                    7393DAD255BEB5751DBDA04AB11CD823BA278C85
```

Now you have the source, and the key, it's enough to check the signature of the last commit:

```
$ git log -n 1 --show-signature
```

The important part is that you get something like this:

```
gpg: Signature made Mon 11 Dec 2017 21:55:28 CET
gpg:                using RSA key 8F7474044095B405D0F042F0A2CCA134BC7C8572
gpg: Good signature from "Luca Fulchir <luker@fenrirproject.org>" [unknown]
gpg:                 aka "Luca Fulchir <luca@fulchir.it>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 7393 DAD2 55BE B575 1DBD A04A B11C D823 BA27 8C85
```

And as long as you got the right key, and you find the **"gpg: Good signature"** string, you can be sure you have the right code.

The main development happens in the *master* branch, while we keep one branch for each stable version.

2.2 Dependencies

libRaptorQ has 4 dependencies:

Eigen3 : This is used for matrix manipulation, which is a big part of RaptorQ.

lz4 : Used to compress cached precomputations.

git : This is used not only to get the source, but also by the build system. We get the last git commit id and feed it to clang or gcc as seed for their internal random number generator. This makes it possible to have reproducible builds.

optionparser : Library used to parse the command line easily in C++, without exceptions.

All dependencies are included in the sources, so you do not need to download and compile them.
Eigen3 is only needed at build time, so there is no need to download it again. we use the 3.3.4 version.
LZ4 is included as a git submodule. if you do not have it, run:

```
$ git submodule init
$ git submodule update
```

and LZ4 w ill be built statically in the library (it will not be installed on the system)

2.3 Build & Install

libRaptorQ uses the cMake build system, so things are fairly standard:

```
$ cd libRaptorQ.git
$ mkdir build
$ cd build
$ cmake ../
$ make -j 4
```

By default, the libRaptorQ project tries to have deterministic builds. This means that if you compile things twice, or with two different computers, the hash of the resulting library will be the same, provided that the same compiler (clang, gcc 4.8, gcc 4.9 etc) was used. Currently the only exception is the clang compiler with the *PROFILING* option enabled, and this will not likely be solved.

There are lots of options, you can use in cmake. As always, you can change them by adding "**-Dcmake_option=cmake_value**" when calling cmake.

You can always use the cmake-gui or ccmake commands to have the list of possible options.

The ones we recognize are:

LTO : ON/OFF. Default:**ON**. Enables *Link Time Optimizatios* for clang or gcc. Makes libraries smaller and better optimized.

PROFILING : ON/OFF. Default:**ON**. Profiling compiles everything once, then runs a test to see which code paths are used more, and then recompiles everything again, but making sure that the binary is optimized for those paths. Works with clang and gcc. Provides a slight speedup.

CMAKE_C_COMPILER : gcc and clang are directly supported. other should work, too. This is only used if you want to build the C example.

CMAKE_CXX_COMPILER : g++, clang++ are directly supported. other should work, too.

CLANG_STDLIB : ON/OFF. Default:**OFF**.use clang's libc++ standard library. Only available with clang.

CMAKE_BUILD_TYPE : Type of build. you can choose between "Debug", "Release", "MinSizeRel" or "RelWithDebInfo"

CMAKE_INSTALL_PREFIX : Default to */usr/local*. Change it to fit your distribution guidelines.

RQ_ENDIANNESS : `Auto,BigEndian,LittleEndian` Force the build to use one endianness ot the other

RQ_LINKER : `Auto,gold,ld,bsd` Force the build to use one linker or the other

USE_LZ4 : ON/OFF. Default:**ON**. compile with support for lz4

Then you can build everything by running:

```
$ make -j 4
```

Of course, you can configure the number of parallel jobs (the *-j* parameter) to be what you need.

Optional make targets: The following optional targets are also supported:

```
$ make docs tests examples
$ make everything
```

The "docs" target builds this document, but you need latex with the refman style. The tests are only useful to check perfromance of rfc compliance right now. "examples" compiles the C and C++ examples, which will not be installed.

Install: The installation process is very simple:

```
$ make install DESTDIR=...
```

You can change the *DESTDIR* parameter to fit your distribution guidelines.

3 Working with RaptorQ

3.1 Theory (you really need this)

To be able to work with liRaptorQ, you must first understand how the RaptorQ algorithms works. We won't go into the details, but just what you need to be able to tune the algorithm to your needs.

Fountain codes: Fountain codes are a special *Forward-Error-Correcting* code class, which characteristic is simple: if you want to send K packets, you actually send $K+X$ packets, and the receiver only needs to get any K packets to be able to reconstruct your data. The number X of overhead packets can be as big as you need (theoretically infinite), so you can tune it to be slightly higher than the expected packet loss.

Systematic codes: RaptorQ is also a systematic code. This means that those first K packets are the input *as-is* (**source symbols**), and the X packets (**repair symbols**) have the information needed to recover any of the lost source packets. This has the big advantage of avoiding any kind of time and memory consuming decoding if there is no packet loss during the transmission.

Complexity: The RaptorQ algorithm is often presented as having a linear time encoder and decoder. This is both false and misleading. Generating the source or repair symbols from the intermediate symbols has linear complexity. Generating the intermediate symbols has cubic complexity on the number of symbols. Which is a completely different thing. It is still very quick. On a core i7, 2.4Ghz, you need to wait *0.4ms* for 10 symbols, *280ms* for 1.000 symbols, but it can take an hour for 27.000 symbols. RaptorQ handles up to 56.403 symbols.

Caching libRaptorQ can work with big matrices that take a lot of time to compute. For this reason the matrices can be saved once they have been computed the first time. libRaptorQ uses a **local** cache. The matrix can be compressed with LZ4.

3.2 RaptorQ: Blocks & Symbols

To understand how to properly use and tune libRaptorQ, you first need to understand how RaptorQ handles its inputs, outputs, and what the time and memory constraints are.

Input sequencing: The RC6330 API needs to have the whole input you want to send before it can start working.
This means that You should buffer the data you are sending, and then when you have a block big enough you pass it to libRaptorQ, which will start generating symbols.

Contrary to the RFC6330, the RAW api is about a single block of data. You will have to divide your input in multiple blocks by hand,

and decide by yourself how to send the encoded symbols. This will increase your flexibility, at the expense of not being conformant to RFC6330. Note that a RaptorQ block does not need to be completely filled since the block is automatically padded with zeros, but you need to somehow tell the receiver how much data you have in that block so that it can know where the padding starts. RFC6330 does this with the `OTI` parameters. **with the RAW API you will have to do it by yourself**. -

Sizes: Each block can have up to 56.403 *symbols*, and each symbol can have up to *size_ t bytes* long (2^{32} on 32 bit machines, 2^{64} on 64 bits). Please remember the Complexity discussion before using high values.

Interleaving: Contrary to the RFC6330 the RAW api does not include any kind of interleaving

Memory and Time: Memory and time requirements are to be considered since RaptorQ needs to run a cubic algorithm on matrix of size $K * K$, where K is the number of symbols in each block.
The algorithm needs to keep in memory two of these matrices, although most of the work is done on only one.
This is actually a lot. More benchmarks and optimizations will come later, for now remember that with 10 symbols it takes something like 0.4ms on a core i7 2.4GHZ, 280ms with 1000 symbols, and up to an hour with 27.000 symbols. YMMV.

3.3 C++ interface

To use the C++ interface you only need to include the **RaptorQ/RaptorQ_v1.hpp** header, and link **libRaptorQ** and your threading library (usually *libpthread*).

You can also use the library as a header-only include, by including **RaptorQ/RaptorQ_v1_hdr.hpp**. All your code should keep working even if you switch between linked and header-only at a later date. If you use the header-only version, remember to have Eigen3 in your include path, and to include the needed definitions, `RQ_LITTLE_ENDIAN` or `RQ_BIG_ENDIAN`

Namespace Since the libRaptorQ library now has two very different usages, everything was split into two namespaces: **RFC6330** and **RaptorQ**. The names should be explanatory: the first namespace is for the RFC interface, while the second namespace is for the low-level interface.

Since we will cover only the **RaptorQ** namespace here, remember that everything we talk about is in this namespace.

Both the RFC6330 and the RaptorQ namespace are versioned so you can keep using old APIs while we change stuff. just add `__v1` to select the first version of the API. E.g: `RaptorQ__v1`

Templates There are two main classes you will use:

```
template <typename Rnd_It, typename Fwd_It>
class Encoder

template <typename In_It, typename Fwd_It>
class Decoder
```

As you might guess, the classes for the encoder and decoder take two template parameters.
For the **Encoder**, the first parameter *MUST* be a *random access iterator* to the input data, and the second parameter is a *forward iterator*. The random access iterator will be used to scan your input data. The forward iterator will be used to write the data to your output.
The same is done for the **Decoder**, but the first iterator can be just an input iterator, and nothing more.

3.3.1 Caching precomputations

libRaptorQ can now cache the most used matrices for a quicker reference. The cache can be scaled up or down dynamically.

local_cache_size : **const uint64_t local_cache**
return: bool
Set the local cache size. returns false on error.

get_local_cache_size() : **return: uint64_t**
get the size of our local cache

supported_compressions() : **return: Compress**
Get the bitmask of all supported compression algorithms. currently only **Compress::NONE** and **Compress::LZ4**.

get_compression() : **return: Compress**
Get the current compression used.

set_compression : **const Compress compression**
return: bool
Try to set a compression algorithm. LZ4 is not a mandatory dependency, so we might not have it

3.3.2 The Encoder

The constructor is the following:

```
Encoder (const Block_Size symbols, const size_t
    symbol_size);
```

Which is really simple. The first is an Enum that will help you get the correct block size, and the second is your arbitrary symbol size.

Since the RaptorQ algorithm only works with specific block sizes, You can get all the possible block sizes from `RaptorQ__v1::*blocks`, which is a pointer to an array of 477 possible sizes, from 10 to 56403. You can cast each element in this array to `uint16_t` to verify the actual value, or directly name them, as in `RaptorQ__v1::Block_Size::Block_42`.

If you pass unsafe values the encoder (or decoder) will refuse to work.

You can instantiate an encoder for example by doing:

```
std::vector<uint32_t> input, output;
...
using T_it = typename std::vector<uint32_t>::iterator;
using RaptorQ = RaptorQ__v1;
RaptorQ::Encoder<T_it, T_it> enc (
                            RaptorQ::Block_Size::Block_42,

                            1500);
```

This will create an Encoder that works on vectors of unsigned 32 bit integers for both input and output, that will be fed to a block of size 42 symbols of size 1500 bytes.

The available methods for the encoder are the following:

operator bool() : **return:bool**
False if constructor parameters did not make sense. Else true.

symbol_size() : **return: uint16_t** The size of a symbol.

symbols : **Input:uint8_t sbn**
return: uint16_t
The number of symbols in a specific block. different blocks can have different symbols.

max_repair : **Input: const uint8_t sbn)**
return: uint32_t
The maximum amount of repair symbols that you can generate. Something less than 2^{24}, but the exact number depends on the number of symbols in a block

begin_source() : **return: Symbol_Iterator<Rnd_It, Fwd_It>**
This returns an iterator to the symbols in this block. Source symbols only

end_source() : **return: const Symbol._Iterator<Rnd_It, Fwd_It>**
This returns an iterator to the end of the source symbols.

begin_repair() : **return: Symbol_Iterator<Rnd_It, Fwd_It>**
This returns an iterator to the symbols in this block. Source symbols only

end_repair : **Input: const uint32_t max_repair**
return: const Symbol._Iterator<Rnd_It, Fwd_It>
This returns an iterator to the end of the source symbols.

has_data() : **return: bool**
Did you feed this encoder the data? That is: did you use `set_data()` without calling `clear_data()` later?

set_data : **Input: const Rnd_It &from**
. **const Rnd_It &to**
return: size_t
Set the iterators from which we load all the data.

clear_data : **return: void**
clear all data, without deallocating memory so that things might be slightly faster next time.

ready() : **return: bool**
Is everything decoded?

stop() : **return void**
Stop any current decoding and return error.

precompute_sync() : **return: bool**
Start a precomputation so that next computations might be slightly faster. Only return when the precomputation is done or stop() was called.

compute_sync() : **return: bool**
Compute internal symbols from the input. Only return when the computation is done or stop() was called.

precompute() : **return: std::shared_future<Error>**
Start a precomputation so that next computations might be slightly faster. Returns immediately, but the user can wait or poll on the returned future.

compute() : **return: std::shared_future<Error>**
Start a computation on the data. Returns immediately, but the user can wait or poll on the returned future.

encode : **Input:Fwd_It &output**
. **const Fwd_It end**
. **const uint32_t id**

return: size_t
Once the computation is finished, you can get the encoded symbols. The first `symbols()` are source symbols, and the next are repair symbols. Returns the number of iterators written into the `output` iterator, which will point to the first non-written element after the symbol.

3.3.3 Symbols

With the *begin_source()/end_source()/begin_repair()/end_repair(..)* calls you get *input iterators* to the symbols. A symbol has the following type:

```
template <typename Rnd_It, typename Fwd_It>
class Symbol
```

and exposes the following methods:

id() : **return: uint32_t**

operator() : **Input: Fwd_It &start**
. **const Fwd_It end**
return: uint32_t
Write the symbol into the start-end iterators provided by the user. Afterwards the start iterator will point to the first non-written element after the symbol

3.3.4 The Decoder

There are two constructors for the Decoder:

```
std::vector<uint32_t> input, output;
using T_it = typename std::vector<uint32_t>::iterator;

RaptorQ__v1::Decoder<T_it, T_it> dec (
                                const Block_Size symbols,
                                const size_t symbol_size,
                                const Report type);
```

The only parameter that is not self-explanatory is the Report type: it regulates when the decoder will tell you that there is available data and will let you extract it.
The possible values are:

- **Report::PARTIAL_FROM_BEGINNING** report the data as it become available, but the data will be sequential
- **Report::PARTIAL_ANY** report any symbol as it becomes available. Could be out-of-order, and you can get symbols before the decoding starts.
- **Report::COMPLETE** wait until the whole block has been decoded.

The decoding methods are:

symbols() : **return: uint16_t**
The number of symbols in this block

symbol_size() : **return: size_t**
The bytes in each symbol

can_decode() : **return: bool**
Whether we have enough data to start decoding or not

ready() : **return: bool**
Whether everything has been decoded or not.

needed_symbols() : **uint16_t**
How many more symbols we need before we try to decode again. If this is 0, you can decode.

clear_data() : **return: void**
clear all data, without deallocating memory so that things might be slightly faster next time.

stop() : **return: void**
Stop any decoding done in the background.

begin() : **return: Symbol_Iterator<In_It, Fwd_It>**
This returns an iterator to the symbols in this block.

end() : **return: const Symbol_Iterator<In_It, Fwd_It>**
This returns an iterator to the end of the symbols in this block

add_symbols : **Input: In_it &from**
. **const In_it &to**
. **const uint32_t esi**
return: RaptorQ__v1::Error
Add one symbol to this block. If the symbol is less than `symbol_size()` the symbol will be padded with zeros

end_of_input : **Input: const Fill_With_Zeros fill**
return: std::vector<bool>
Tell the decoder that we know that there will be no more data for this block. The first parameter is an `enum`, either **Fill_With_Zeros::YES** or **Fill_With_Zeros::NO** If you choose `YES` then the repair symbols will be discarded, the missing symbols will be filled with zeros and for output you will get the bitmask for which byte was received (true) or was filled with zeros (false).
If you choose **Fill_With_Zeros::NO** you will get *an empty vector*, and the decoding will end with error if it was waiting for more input.

set_max_concurrency : **Input: const uint16_t max_threads**
return: void
The decoder can try to decode the same block multiple times, while more repair symbols arrive. Unless you really need it, leave the default, which is 1, so no decoding concurrency or the same block.

decode_once() : **return: RaptorQ__v1::Decoder_Result**
Try to decode the block, only return once the decoding is finished, do not try again even if more repair symbols arrived.

poll() : **return: RaptorQ__v1::Decoder_wait_res**

```
struct Decoder_wait_res {
    Error error;
    uint16_t symbol;
};
```

In general, the `error` can be **Error::WORKING**, `Error::NEED_DATA`, `Error::NONE`. The last one signals that the block (or symbol, see later) has been decoded. The `symbol` parameter changes depending on the **RaptorQ__v1::Report** type that you set when you constructed the decoder:

- **Report::PARTIAL_FROM_BEGINNING**: `symbols` will be the number of symbols available, but counting only the sequential ones starting from the beginning

- **Report::PARTIAL_ANY**: `symbols` will be the first symbol available (if `error` == `Error::NONE`). If the block has been decoded `symbols` == `decoder.symbols()`
- **Report::COMPLETE** `symbols` is not used.

wait_sync() : **return: RaptorQ__v1::Decoder_wait_res**
Start the decoding, and keep trying or waiting for new data to arrive until either we successfully decode the block or `stop()` is called. Also stops waiting for data after `end_of_input` is called (but will keep retrying if there already is enough).
The result is the same as above

wait() : **return: std::future<RaptorQ__v1::Decoder_wait_res>**
Same as before, but returns a future immediately. This call is enabled only if you are compiling with C++11 or later

decode_symbol() : **Input: Fwd_It &start**
. **const Fwd_It end**
. **const uint16_t esi**
return: RaptorQ__v1::Error
Decode a single symbol. If $end - start$ is less then the symbol size, then you will get a partial symbol. **return: RaptorQ__v1::Error**

decode_bytes : **Input: Fwd_It &start**
. **const Fwd_It end**
. **const size_t from_byte**
. **const size_t skip**
return: Decoder_written
Decode a specific range of the block. The `skip` parameter is the offset (in bytes) to skip from the `size` iterator. This is done to allow to work with iterators whose data is bigger than 1 byte. Although we recommend to work with uint8_t data.
Returns the number of bytes written, offset of the data in the last iterator (0 if it is aligned to the iterator)

```
struct RAPTORQ_API Decoder_written {
    size_t written;
    size_t offset;
};
```

4 GNU Free Documentation License

GNU Free Documentation License
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful 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, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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 fewer than five), unless they release you from this requirement.

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", Preserve 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. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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, is called an "aggregate" if the copyright
resulting from the compilation is not used to limit the legal rights
of the compilation's users beyond what the individual works permit.
When the Document is included in an aggregate, this License does not
apply to the other works in the aggregate which 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 half of
the entire aggregate, the Document's Cover Texts may be placed on
covers that bracket the Document within the aggregate, or the
electronic equivalent of covers if the Document is in electronic form.
Otherwise they must appear on printed covers that bracket 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, and all the license notices in the
Document, and any Warranty Disclaimers, provided that you also include
the original English version of this License and the original versions
of those notices and disclaimers. In case of a disagreement between
the translation and the original version of this License or a notice
or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements",
"Dedications", or "History", the requirement (section 4) to Preserve

its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this

License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

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 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other
combination of the three, merge those two alternatives to suit the
situation.

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.