MuLan-String library manual

Przemysław Chojnacki

March 19, 2022

Contents

1 Introduction

MuLan-String (or MultiLanguage String) is an internationalization and localization library made to aid translators whose native language nature forbids simple strings substitution eg. because it uses case system. The other goal of MuLan-String is to make working with such languages simple for programmers writing programs with i18n in mind.

The general idea of using MuLan-String is based on *template strings*. The program using the library uses them for building strings from elements. The translators formulate template strings which can include rules of how to generate text based on language's case system, genders or pluralization rules.

This manual shows how to use MuLan-String from the view point of programmer and translator. One can read only the part made for one point of view and skip the other.

2 The programmer's POV

MuLan-String was made to be easy to use in source code in different projects. Its main parts are the template system and the backend. Both parts are made with customization in mind. To use the library one have to "install" it in a project. To install the library you have to choose one of the files included with this manual:

`mulanstring.hpp` **file** contains only the core of the MuLan-String library. You can use this file, if you want to use your own backend.

`mulanstring-gettext.hpp` **file** contains also the GNU Gettext backend bundled with the MuLan-String. If your project's internationalization system is based on GNU Gettext library, you should choose this file

The installation itself is made around the idea of one file being both a header and an implementation[1]. For every source file in your project that requires MuLan-String you have just put a normal `#include` directive pointing to the file you have chosen from the above list.

```
#include "mulanstring.hpp"
```

Then you have to choose *one file* in your project that will contain the implementation like so:

```
#define MULAN_STRING_IMPLEMENTATION
#include "mulanstring.hpp"
```

If you want to use MuLan-String with default options that's all you need. You are ready!

2.1 Common options

The previous section has shown the basic installation. Sometimes, however, you may want to change some tune up the library more to your needs. MuLan-String offers you the bellow adjustments:

- How to treat invalid templates,
- What are the template markers,
- (For GNU Gettext) if the library should include underscored functions

Invalid templates: The first decision you may want is to choose how MuLan-String should react on improperly formatted template strings. By default a wrong template object produces an empty string, no matter how it's called. If, for some reason, you want the MuLan-String to throw an error you have to put a `#define` directive in the file that's including the library's implementation:

```
#define MULANSTR_THROW_ON_INVALID_TEMPLATE
#define MULAN_STRING_IMPLEMENTATION
#include "mulanstring.hpp"
```

[1] You may know it under the name "STB library"

Template markers: Another decision you may want to make is what delimiters are used in template strings. By default in MuLan-String templates are made using `%{` and `}%` markers. For example in:

> %{parent}% has %{num}% kids

template the `%{parent}%` and `%{num}%` are substitution commands. If for some reason you want to use `[[` and `]]` as delimiters, you have to write:

```
#define MULANSTR_TAG_START "[["
#define MULANSTR_TAG_END "]]"
```

in the implementation file (before the `#include` directive).

Some template tags needs parameters. For example in:

> There is %{num}% file%{num!P one={} other={s}}%

template the P function returns text based of the quantity given in the `num` variable. It then produces output based on whenever `num` is 1 or >1 (for English language). So, the above example will result in "There is 1 file" or "There is 8 files" (for `num` being 1 and 8 respectively). For this it needs to know what text to output. And these texts are given between `{` and `}` delimiters.

If you want to change those internal delimiters to `<` and `>`, you have to add:

```
#define MULANSTR_INNER_TAG_START "<"
#define MULANSTR_INNER_TAG_END ">"
```

to the above list.

Helper functions: The other option is to tell the library if we want to use 2 helper functions which all start with an underscore. These are meant to speed up writing programs. They are short name replacements for functions retrieving template strings from the backend. If you don't want them just write:

```
#define MULANSTR_DONT_USE_UNDERSCORE
```

in the known location.

These functions (and their replacements) are:

1. `_(message)` or `mls::translate(message)`: the basic template retriever. Gets template in the default language set during initialization.
2. `_c(catalog, message)` or `mls::translate(catalog, message)`: Gets template from a different catalog. In the GNU Gettext 'catalog' is the name of a '`.mo`' file.

2.2 Working with backends

MuLan-String's templates are taken from a backend. Different backends require different method to extract translatable strings from your project's code. Another fact is some backends require an initialization process before they can retrieve strings. How to do it is explained below.

2.2.1 GNU Gettext

To initialize Gettext you need to call `mls::backend::init(...)` function. This function requires a name of the main catalog used by our program. Usually it is the project name set by `#define PACKAGE ...` in the `config.h` header (if you are using Autoconf). Other parameters are: the locale name and localization of `.mo` files. These are optional and, if not provided, are set to their default values (which for the locale is the system locale and for the localization is `/usr/local/share/locale`). The initialization should be done as earlier as possible, preferably in the `main()` function:

```
int main() {
// ...
mls::backend::init(PACKAGE);
//or if you want to set locale:
mls::backend::init(PACKAGE, "en_US");
//or if you want to set localization:
mls::backend::init(PACKAGE, nullptr, "./locales");
//or all three:
mls::backend::init(PACKAGE, "en_US", "./locales");
// ...
}
```

To extract template strings you have to run `xgettext` with these parameters:

`xgettext -C -k_ -k_c:2 -o` <**name of the output .pot file**> <*list of .hpp and .cpp files*>

or, if you have set `MULANSTR_DONT_USE_UNDERSCORE` you need to write a longer list:

`xgettext -C -kmls::translate -kmls::translate:2 \`
`-o` <**name of the output .pot file**> <*list of .hpp and .cpp files*>

For information on how to work with `.pot`, `.po` and `.mo` files refer to the GNU Gettext manual[2].

2.3 Using MLS templates in code

The idea of MuLan-String is based around template strings. You use the library by making objects of the `mls::Template` class. How to do it depends on if you preferred to use some backend or not:

2.3.1 No backend

This solution gives you the greatest freedom over how templates are constructed. It also may be the least comfortable, as you have to provide also a *locale* for every template object. To do so, call the `mls::locale::getLocale(...)` function:

```
auto myLocale = mls::locale::getLocale("en_US");
```

(the locale names are listed in the Supported languages section.)

To obtain a template object you have to make a `mls::Template` variable with *template string* and *locale* object passed to its constructor:

```
mls::Template aTemplate{"Template string", myLocale};
```

2.3.2 GNU Gettext backend

Having GNU Gettext as our backend, you can get it by using special template retrieval functions discussed in the 2.1 subsection. You can do it as in the example:

```
auto filesFound = _("%{num}% file%{num!P:,s}% have been found");
```

This will give you template object with locale set during the initialization.

2.3.3 Applying variables

Some templates require additional information to produce a result string. In the above example, the template needs `num` to be set for the number of files found. We give that information by invoking `apply(<var name>, <value>)` method on the template:

```
filesFound.apply("num",3);
```

[2] Available at https://www.gnu.org/software/gettext/manual/index.html

Some templates have got more than one variable. To set them, we call `apply(...)` for each of the variables. You can do it by separate calls or chain them:

```
auto parentHasKids = _("%{parent}% has got %{num}% %{num!P:kid,kids}%");

//1st method
parentHasKids.apply("parent", "Alice");
parentHasKids.apply("num", 3);
//2nd method
parentHasKids.apply("parent", "Alice").apply("num", 3);
```

2.3.4 Producing output

After getting and applying variables (if necessary) we can get the result by calling `get()` method on the template. This method produces a `std::string` which can be used elsewhere in the program.

```
std::cout << parentHasKids.get() << std::endl;
```

2.3.5 All three in one line

All that was said above can be written in one line:

```
std::cout << _("%{parent}% has got %{num}% %{num!P:kid,kids}%")
.apply("parent", "Bob").apply("num", 2)
.get() << std::endl; //produces "Bob has got 2 kids"
```

2.4 The `apply` family

What can you put in the `apply(...)` method? A couple of things:

Numbers You can set a template's variable to a number by using the `apply(varName, number)` method for integers up to the `long` size, or `applyReal(varName, real)` for real numbers up to the `double` size. Numbers can be used for their raw value or as an input to some functions.

Strings Raw strings are printed as-is, without any formatting done to them. They should be used for anything that must not be or can not be modified by a translator. Use them sparingly as this blocks your translator from applying any of MuLan-String powerful functions!

Other templates When you want to insert some text into another text, then this is the right way of doing it. It let's your translator to use the full potential of the MuLan-String template system. As a rule of thumb you should choose this method, if you want to put single words inside a sentence. These words should be made in their own template objects and retrieved from your backend.

If, for some reason, you can't do that (because, for example, these words are generated on-the-fly or come from some external source), you should fallback to raw strings. It would be then a good thing if you let your future translators get to know if this is the case (for example by using MuLan-String comments, see the next section).

2.5 A quick overview of the template string syntax

So far now I told you how to get and work with templates, but nothing about *what to put in them*. This is the job of this subsection. However, I won't show all the possibles of the MuLan-String library. The reason is you as the programmer are using English in the source code and as the default language for communicating with a user. So, you only need to know those elements of MuLan-String's template system which are enough to work in English. If you want to know more, read the Translators' POV section of this manual.

OK, so what MuLan-String's templates are made of? The templates are strings with tags which may be identified by `%{` and `}%` delimiters[3]. Everything inside them tells the MuLan-String to do its "magic".

There are two main types of tags: a substitution tag and a function tag:

[3] Remember, you can change the delimiters

Substitutions are tags in the form `%{`*variable_name*`}%`. They work by putting a text (or a number) from a variable named the same as it is written between the delimiters. The content of that variable are given by the `apply(...)` method(s) of a template.

Functions are tags in the form `%{`*variable_name*`!FUNCTION` *arguments...*`}%`. They work in the similar way to the above, but use one of MuLan-String functions to process the content of the *variable_name* in some way and produce the output based on its result. For the English users the only functions you must know about are:

- the `P` function, which stands for *(P)luralize*,
- the `I` function that prints and formats an *(I)nteger*,
- the `R` function that prints and formats a *(R)eal number*.

2.5.1 Pluralizer

The function gets two parameters and a variable name. The parameters tells the function what to output if the variable is equal to 1 or not.

The arguments may be given in two possible ways:

- **As a table**: `%{`*variable_name*`!P:`*singular form*`,`*plural form*`}%`
- **As a hash**: `%{`*variable_name*`!P one={`*singular form*`} other={`*plural form*`}}%`[4]

Example:

> `%{num}%` `%{num!P:page,pages}%` `%{num!P one={has} other={have}}%` been printed.

Suppose we have set *num* to be equal 1. The first tag in the above template will put the string "`1`" in its place. The second tag will check the *num* and, because it is equal 1, it will put the "`page`" string in its place. The third tag works the same way, but its argument list was given in a more verbose form. The tag in this case will get the string given in the `one` parameter and put the "`has`" string in its place.

As a result the output will be: `"1 page has been printed."`.

Now, let's suppose the *num* is equal to 4. The first tag will produce "`4`", the second: "`pages`" and the third: "`have`" which results in an output: `"4 pages have been printed."`.

Take note that the above example may be written also in this way:

> `%{num}%` page`%{num!P:,s}%` ha`%{num!P one={s} other={ve}}%` been printed.

The produced result will be the same. It's up to you which form you think is more readable and maintainable.

2.5.2 Integer formatter

The next function in your (English) toolset is `I`. It is a simple function that prints an integer (and only an integer). What makes it different from simple `%{var}%` substitution? Well, it allows you (and your translator) to let your number string follow the formatting rules of an language. In English language it boils down to add "," characters in numbers bigger than a thousand.

You use this function by giving it an argument in this way:

- `%{`*variable_name*`!I=general}%` or
- `%{`*variable_name*`!I=grouped}%`

The first version prints number more of less as-is without grouping. The second one adds grouping characters when necessary.

[4] Remember you can change internal { and } characters to anything else in your setup

Example:

Your big number is `%{num!I=general}%` or `%{num!I=grouped}%` (pretty printed).

Now if you set the *num* variable to a million it will produce the string:

Your big number is 1000000 or 1,000,000 (pretty printed).

What to use depends on your business requirements.

Side note: Are there any other formats besides "general" and "grouped"?
You may wonder why the `I` function requires an argument. Why to bother if one can simply print numbers with `%{num}%` and occasionally use `%{num!I}%` for cases when it requires the number to be pretty formatted? Well, one case that you might consider is printing *money* amounts. This is one moment when you might think it would be enough to just print a number with thousand's separators like anywhere else (adjusted to the specific language's separator character).
Unfortunately this way of thinking will let your *Swiss* users get angry at you. In this country there are different formatting rules for printing *regular* numbers and the mentioned *money amounts* numbers. Normal numbers are formatted in this way:
`1 234 567,89`
And money are printed like that:
`1'234'567.89`
so forcing users of your application to chose only between two number formats may be not as well thought idea as you may think.

2.5.3 Real numbers formatter

The `R` function is used in a similar way as the `I` function. It let's you print numbers with fractional parts in a way that is compliant with the rules of the destination language (like for example using the right fraction separator character).

You can use this function in one of the below ways:

- `%{num!R:`*number format*`}%`
- `%{num!R:`*number format*`,`*precision*`}%`
- `%{num!R format={`*number format*`}}%`
- `%{num!R prec={`*precision*`}}%`
- `%{num!R format={`*number format*`} prec={`*precision*`}}%`

As you can see you may pass the arguments in both *short* and *verbose* format and specify the required *precision* as well. If you don't write precision, the MuLan-String assumes you want to print a number with as many fractional digits as it is necessary.

Example:

The Earth's radius is equal to `%{radius!R:grouped,3}%` km.

This will result in a string:

The Earth's radius is equal to 6 356.752 km.

2.5.4 Comments

The last thing to know about MuLan-String templates is you can put comments inside them. They are made in this way:

Text `%{#`*comment*`#}%` around.

What MuLan-String does with them is it treats them as if they weren't there. So, the result of the above example will be:

`Text␣␣around.`

(mind the double spaces between words!)

What comments are useful for? Well, sometimes to translate the sentence translators need some external information. However the process of extracting templates from a source code may remove that information. Let's suppose we want translators to translate "Sam is beautiful." In some languages the word "beautiful" will be different, depending on whenever Sam is a man or a woman. But all that a translator sees is just that simple text. It would be good, if we had some way to inform our translators about the Sam's gender, thus solving the ambiguity.

And that's is why comments may be useful. You can write the above example as:

Sam`%{#a woman#}%` is beautiful.

3 The translators' POV

As a translator the only thing you must learn is the MuLan-String's template syntax. The syntax is based on a system of tags inserted inside template string. They are recognized by being surrounded by delimiters. These delimiters are, by default, `%{` and `}%`. So, for example the below sentence:

> You have selected `%{num}%` `%{num!P:object,objects}%` for deletion.

contains two tags: `%{num}%` and `%{num!P:object,objects}%`. The first tag is called *the substitution tag* and the second: *the function tag*. Everything that is surrounded by these delimiters is subject to MuLan-String "magic."

Warning! The programmer has the ability to change those delimiters to anything else! Make sure you consult the programmers' team in case you would suspect they have done it.

3.1 Types of tags

OK, so what types of tag content you may encounter/use? Well, at first let me tell you about general types of tags:

3.1.1 Substitution tags

This type is the simplest of all and also the most restricted in how it may behave. They are recognized by one word put between delimiters:

> `%{`*variable*`}%`

All what it does is to put the *variable* content (given by a programmer) in its place. You, as the translator, must make sure that content will be put in the right place. You do it by moving such a tag into a place where it is the most logical for a sentence in your language.

So, if given a template:

> There were `%{num}%` changes in the `%{document_type}%`

If your language for some reason requires that *document_type* should be to put before *num*, you can switch those two tags like that:

> In `%{document_type}%` there is `%{num}%` changes.

The order of them doesn't matter as long as all of them are in place.

3.1.2 Function tags

Ah, there is where MuLan-String gets its power! This tag type returns the result of a call to one of predefined functions. These functions works by getting the variable name (or not) and some parameters and, based on that information, produces an output.

A function can take one parameter, a table, or a hash (also called a map). Also some functions needs a variable name to work, others doesn't need it. If that sounds complicated, don't worry: I explain all of it in parts.

The first thing is whenever the function needs a variable or not:

Requred: functions of this type are written in this form:

> `%{`*variable_name*`!FUNCTION_NAME`*argument(s)*`}%`

(mind the `!` character that separates the variable name from function name)

Not needed: this type of functions are written in this form:

> `%{+FUNCTION_NAME`*argument(s)*`}%`

(remember about the plus sign at the beginning of the function name)

The second thing is how a function gets its arguments. An argument list goes right after the function name, and can take one of these forms:

Form	Description	Example
`=`*value*	gives *one* parameter	`%{item!C=gen}%`
`:`*first*`,`*second*`,...`	gives *a table*	`%{num!P:mouse,mice}%`
␣`par1=`*value1*␣`par2=`*value2*`...`	gives *a map*	`%{person!G m={his} f={hers}}%`

I hope all of it will be clear once you read the subsection about functions.

3.1.3 Comment tags

And the last are tags which serve as an aid for translators given by programmers. Sometimes you may find such that tag:

`%{#`*a commentary*`#}%`

In a translation you can do anything to that tag, even remove it completely.

What is their purpose? Well, sometimes you may find a template which is hard to translate without some external information. For example a text:

A kid just ran away.

may be hard or even impossible to translate, if in your language "ran away" requires the knowledge about the kid's sex. You then may ask a programmer to clarify it by putting a comment like this:

A `%{#male#}%`kid just ran away.

Another reason is some backends like GNU Gettext may remove duplicate templates which, for some cases, can make a problem for a translator. Let's imagine we got this template:

Open

and an information that it is used in both "File" and "Print" menus. For some languages the translation of "Open" may differ depending on if it relates to a file or a printer. Unfortunately, your backend may treat the second "Open" as redundant, leaving you with a problem how to write a translation which fits both cases.

The solution again lies in the possibility for programmer to add comments. For example:

```
%{#File menu#}%Open
%{#Print menu#}%Open
```

will not be reduced to one item and gives a translator a way to translate them differently.

3.2 List of functions

OK, so after we have learned how to write function tags, now I can tell you what possibilities MuLan-String gives to you. I divided the list into 4 parts by the topic. Each topic tells about problems you may encounter during a translation.

3.2.1 Gender

Gender tells about a class a word belongs to. It is a peculiarity of each noun and a language, which have got this feature, may require other words to adapt to that other word.

MuLan-String gives two functions to work with genders:

(S)et (G)ender ⇒ `%{+SG=`*gender_to_set*`}%`

Each template has got a gender assigned to them. Normally it is an empty string, but using the SG function we can change that.

This function produces nothing in place where it was put, so alone it is not useful. But becomes one when used with the next function.

(G)ender writer ⇒ `%{noun!G:`*a list*`}%` **or** `%{noun!G` *a map*`}%`

Produces output based of gender set to the *noun*. To work right the value of *noun* variable must be another template where SG function was used.

Example: Let's say we have a template (written in Latin):

`%{noun}%` magn`%{noun!G m={us} f={a} n={um}}%` est

and three other templates:

`%{+SG=m}%`Puteus
`%{+SG=f}%`Officina
`%{+SG=n}%`Forum

If now a program sets the *noun* variable to one of those three templates, the results will be:

`Puteus magnus est` (for *noun* set to the first template)
`Officina magna est` (for the second template)
`Forum magnum est` (and for the third template)

Short and long version In the example above the G function was written in the long (verbose) form. When executed it tries to match each of the given parameters names (`m`, `f` and `n`) to those set by the SG function in the sub-template. Sometimes however you don't want to write it *that* long and want some shorter and quicker form to write. As you can see in the function header in this manual, the G function has a shorter form: `%{noun!G:`*a list*`}%`. To use it in our Latin example we need to replace it with:

`%{noun}%` magn`%{noun!G:us,a,um}%` est

Now the *-us*, *-a* and *-um* endings are assigned to the `m=`, `f=` and `n=` parameters in order. The order of that parameters depends on the language. The MuLan-String has an embedded list of languages it supports. Each entry in that list contains, among others, a specific order set to the Gender writer function, which dictates how a list is converted into a map. You can read what is that order in the Supported languages section.

3.2.2 Case

Some languages employ a case system. A form of a noun depends of what a function it has in a sentence. If, for example, a noun is used as sentence's subject it has different form than when used as an object.

For this feature MuLan-String has got two functions:

(C)ase chooser ⇒ `%{noun!C=`*required_case_name*`}%`

Outputs the *noun* variable, first informing it it should inflect by *required_case_name*.

(C)ase writer ⇒ `%{+C case1={`*case1_output*`} case2={`*case2_output*`}}%` **or** `%{+C:`*list of outputs*`}%`

If a template containing this function was informed by the Case chooser to inflect, this function outputs a `case`*n*`_output` associated with `case`*n*.

Example: Again a Latin example. There is a template for a word *domus* (house):

dom%{+C nom={us} gen={us} dat={ui} acc={um} abl={o} voc={us} loc={i}}%

and a template:

De %{from!C=abl}% in %{to!C=acc}%

which means "From *from* to *to*".

If now the program assigns both variables to the same "domus" template, the output will be:

De **domo** in **domum**

correctly inflected.

Short and long version just like in the Gender section, the Case writer also has got a long and a short form. The short form of the "domus" template may be like:

dom%{+C:us,us,ui,um,o,us,i}%

And just like for Gender functions, the order of assigning elements of that list to a map is given in the Supported languages section.

3.2.3 Plural forms

Pluralization rules tells how a noun should change its form based of some numerical quantity. Languages vary *a lot* in this matter.

There is only one function for this feature in MuLan-String.

(P)luralizer ⇒ %{num!P one={*output_for_1*} other={*output_for_the_rest*}}% **or** %{num!P:*list of outputs*}%

This function gets a number from the variable and gives an output depending of the value and pluralization rules in the template's language.

The Supported languages section contains the rules, list of classes of numbers and how short form of the function is mapped to the long form.

Example: In the English language nouns has got singular and plural forms. Singular forms are applied to quantities equal to 1 and plural forms for the rest. Therefore one can write a template like:

%{num}% file%{num!P one={} other={s}}% deleted

Now if a program sets *num* variable to 1 the output will be:

1 file deleted

and with $num = 3$ it will be:

3 files deleted

3.2.4 Number formats

Functions in this section operates on numbers. They are meant to print numbers with the right thousands separator and fractional separator. Also these functions take care of separating numbers in groups with the right amount of digits in each group.

There are two functions in this group:

(I)nteger formater ⇒ %{num!I=*number format name*}%

It prints an integer (a number without any fractional parts). It takes one required parameter: *number format name*. To know what are the names, refer to the Supported languages section. However, mostly you can put there one of those values:

general It (usualy) prints a number without any separators.

grouped Prints a number grouped and separated.

What to use depends on what you think should be acceptable in your language in a specific template case. You are free to change the programmer's decision.

(R)eal number formater ⇒ `%{num!R:` *a list*`}%` or `%{num!R` *a map*`}%`

This one is slightly more complicated function. It prints a real number (a number with some fractional parts). It takes as parameters a number format name (used like in the previous function) and optionaly a precision.

It is used in one of these ways:

- `%{num!R:`*number format*`}%`
- `%{num!R:`*number format*,*precision*`}%`
- `%{num!R format={`*number format*`}}%`
- `%{num!R prec={`*precision*`}}%`
- `%{num!R format={`*number format*`} prec={`*precision*`}}%`

Example: Let's consider the template:

> You owe to the bank $`%{amount!R format{grouped} prec={2}}%`.

When printed in English it may result in:

> You owe to the bank $1,234.56.

However, if you translate the above template to some of a Swiss bank application, you may write it like so:

> Sie schulden der Bank `%{amount!R format={money} prec={2}}%` CHF.

which may result in this string (after doing the right variable substitution):

> Sie schulden der Bank 1'234.56 CHF.

Take note of one thing: the programmer have chosen a *grouped* number format, because that's the right format used when printing an amount of money in English. However you, a supposedly Swiss translator, have changed that choice to special *money* number format, that formats the value as it should be formatted in the Swiss language with regard to money.

3.3 Supported languages

This section contains the list of all languages the MuLan-String has a builtin support.

British English

Locale name: `en_GB`

Cases list: *None*

Genders list: *None*

Plurals list: one($= 1$), other($\neq 1$)

Number formats: general, grouped (#, ###, ###.00)

American English

Locale name: `en_US`

Cases list: *None*

Genders list: *None*

Plurals list: one($= 1$), other($\neq 1$)

Number formats: general, grouped (#, ###, ###.00)

Polish

Locale name: `pl_PL`

Cases list: nom, gen, dat, acc, ins, loc, voc

Genders list: m, f, n

Plurals list: one($= 1$), few($ending = [2, 3, 4]$ except $ending = [12, 13, 14]$), other

Number formats: general, grouped (# ### ###,00)

A 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 https://www.gnu.org/licenses/.

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.