Muse manual

an authoring and publishing environment
for GNU Emacs and XEmacs

This manual is for Emacs Muse version 3.20.

Table of Contents

1 About the documentation

This document describes Muse, which was written by John Wiegley and is now maintained by Michael Olson. Several versions of this manual are available on-line.

- PDF: http://mwolson.org/static/doc/muse.pdf
- HTML (single file): http://mwolson.org/static/doc/muse.html
- HTML (multiple files): http://mwolson.org/static/doc/muse/

2 What is Muse?

Emacs Muse (also known as “Muse” or “Emacs-Muse”) is an authoring and publishing environment for Emacs. It simplifies the process of writing documents and publishing them to various output formats.

Muse consists of two main parts: an enhanced text-mode for authoring documents and navigating within Muse projects, and a set of publishing styles for generating different kinds of output.

What makes Muse distinct from other text-publishing systems is a modular environment, with a rather simple core, in which "styles" are derived from to create new styles. Much of Muse’s overall functionality is optional. For example, you can use the publisher without the major-mode, or the mode without doing any publishing; or if you don’t load the Texinfo or LaTeX modules, those styles won’t be available.

The Muse codebase is a departure from emacs-wiki.el version 2.44. The code has been restructured and rewritten, especially its publishing functions. The focus in this revision is on the authoring and publishing aspects, and the "wikiness" has been removed as a default behavior (available in the optional ‘`muse-wiki`’ module). CamelCase words are no longer special by default.

One of the principal aims in the development of Muse is to make it very easy to produce good-looking, standards-compliant documents.

3 How to Get Muse Releases and Development Changes

3.1 Released versions of Muse

Choose to install a release if you want to minimize risk.

Errors are corrected in development first. User-visible changes will be announced on the `muse-el-discuss@gna.org` mailing list. See Chapter 12 [Getting Help and Reporting Bugs], page 59.

Debian users can get Muse via apt-get. The '`muse-el`' package is available both at Michael Olson's APT repository and the official Debian repository. To make use of the former, add the following line to your '`/etc/apt/sources.list`' file and run `apt-get install muse`.

```
deb http://mwolson.org/debian/ ./
```

Ubuntu users can also get Muse via apt-get. The '`muse-el`' package is available both at Michael Olson's APT repository and the official Ubuntu repository. To make use of the former, add the following line to your '`/etc/apt/sources.list`' file and run `apt-get install muse`.

```
deb http://mwolson.org/ubuntu/ ./
```

The reason for making separate Debian and Ubuntu packages is that this manual is under the GFDL, and Debian will not allow it to be distributed in its main repository. Ubuntu, on the other hand, permits this manual to be included with the '`muse-el`' package.

Alternatively, you can download the latest release from `http://download.gna.org/muse-el/` .

3.2 Latest unreleased development changes

Choose the development version if you want to live on the bleeding edge of Muse development or try out new features before release.

The git version control system allows you to keep up-to-date with the latest changes to the development version of Muse. It also allows you to contribute changes (via commits, if you are have developer access to the repository, or via patches, otherwise). If you would like to contribute to Muse development, it is highly recommended that you use git.

If you are new to git, you might find this tutorial helpful: `http://www.kernel.org/pub/software/scm/`

Downloading the Muse module with git and staying up-to-date involves the following steps.

1. Install git.
   - Debian and Ubuntu: *`apt-get install git-core`*.
   - Windows: `http://git.or.cz/gitwiki/WindowsInstall`.
   - Other operating systems: download, compile, and install the source from `http://www.kernel.org/pub/software/scm/git/`, or find a git package for your operating system.
2. Download the Muse development branch.

   If you have developer access to Muse, do:

```
git clone ssh://repo.or.cz/srv/git/muse-el.git muse
```

otherwise, do:

```
git clone git://repo.or.cz/muse-el.git muse
```

If you are behind a restrictive firewall, and do not have developer access, then do the following instead:

```
git clone http://repo.or.cz/r/muse-el.git muse
```

3. List upstream changes that are missing from your local copy. Do this whenever you want to see whether new changes have been committed to Muse. If you wish, you may skip this step and proceed directly to the "update" step.

```
# Change to the source directory you are interested in.
cd muse

# Fetch new changes from the repository, but don't apply them yet
git fetch origin

# Display log messages for the new changes
git log HEAD..origin
```

"origin" is git's name for the location where you originally got Muse from. You can change this location at any time by editing the '`.git/config`' file in the directory where the Muse source was placed.

4. Update to the latest version by pulling in any missing changes.

```
cd muse
git pull origin
```

git will show how many files changed, and will provide a visual display for how many lines were changed in each file.

There are other ways to interact with the Muse repository.

- Browse git repo: `http://repo.or.cz/w/muse-el.git`
- Latest development snapshot: `http://mwolson.org/static/dist/muse-latest.tar.gz`
- Latest development snapshot (zip file): `http://mwolson.org/static/dist/muse-latest.zip`

The latest development snapshot can lag behind the git repo by as much as 20 minutes, but never more than that.

Becoming a Muse developer

If you want commit access to the shared Muse repository, then register an account at `http://repo.or.cz` (be sure to add an SSH key), and contact the current maintainer at `mwolson@gnu.org`. It would be best to send some patches to the `muse-el-discuss@gna.org` mailing list first, so that he knows that you know what you are doing. See Chapter 12 [Getting Help and Reporting Bugs], page 59, for instructions on subscribing to the mailing list.

You must also be willing to sign a copyright assignment for your changes to Muse, since Muse is a GNU project. The current maintainer will assist you in this process if you contact him.

For information on committing changes to Muse and performing development, please consult `http://emacswiki.org/cgi-bin/wiki/MuseDevelopment`.

4 Compiling and Installing Muse

Muse may be compiled and installed on your machine.

Compilation

This is an optional step, since Emacs Lisp source code does not necessarily have to be byte-compiled. Byte-compilation may yield a very slight speed increase.

A working copy of Emacs or XEmacs is needed in order to compile Emacs Muse. By default, the program that is installed with the name `emacs` will be used.

If you want to use the `xemacs` binary to perform the compilation, you must copy '`Makefile.defs.default`' to '`Makefile.defs`' in the top-level directory, and then edit '`Makefile.defs`' as follows. You can put either a full path to an Emacs or XEmacs binary or just the command name, as long as it is in the `PATH`.

```
EMACS    = xemacs
SITEFLAG = -no-site-file
# Edit the section as necessary
install_info = install-info --section "XEmacs 21.4" $(1).info \
        $(INFODIR)/dir || :
```

Running `make` in the top-level directory should compile the Muse source files in the '`lisp`' directory, and generate an autoloads file in '`lisp/muse-autoloads.el`'.

Installation

Muse may be installed into your file hierarchy by doing the following.

Copy '`Makefile.defs.default`' to '`Makefile.defs`' in the top-level directory, if you haven't done so already. Then edit the '`Makefile.defs`' file so that `ELISPDIR` points to where you want the source and compiled Muse files to be installed and `INFODIR` indicates where to put the Muse manual. You may use a combination of `DESTDIR` and `PREFIX` to further determine where the installed files should be placed. As mentioned earlier, you will want to edit `EMACS` and `SITEFLAG` as shown in the Compilation section if you are using XEmacs.

If you are installing Muse on a Debian or Ubuntu system, you might want to change the value of `INSTALLINFO` as specified in '`Makefile.defs`'.

If you wish to install Muse to different locations than the defaults specify, edit '`Makefile.defs`' accordingly.

Run `make` as a normal user, if you haven't done so already.

Run `make install` as the root user if you have chosen installation locations that require root permissions.

ELPA

For those used to installing software packages, there will be a `muse` package available in the Emacs Lisp Package Archive (abbreviated "ELPA") as of the 3.10 release of Muse. This package will be compiled and installed automatically in a user-specific location. For more information on ELPA, see `http://tromey.com/elpa/`.

5 Getting Started

5.1 How to Load Muse

To use Muse, add the directory containing its files to your `load-path` variable, in your '`.emacs`' file. Then, load in the authoring mode, and the styles you wish to publish to. An example follows.

```
(add-to-list 'load-path "<path to Muse>")

(require 'muse-mode)     ; load authoring mode

(require 'muse-html)     ; load publishing styles I use
(require 'muse-latex)
(require 'muse-texinfo)
(require 'muse-docbook)

(require 'muse-project)  ; publish files in projects
```

An easy way of seeing which settings are available and changing settings is to use the Muse customization interface. To do this, type *M-x customize-group muse RET*. Each of the options has its own documentation. Options are grouped logically according to what effect they have.

5.2 How to Edit Files in Muse

Muse Mode should automatically be activated when you visit a file with a ".muse" extension. One such file is '`QuickStart.muse`', which is available in the '`examples`' directory of the Muse distribution. You can tell that Muse Mode has been activated by checking for the text "Muse" in your mode line. If Muse Mode has not been activated, you may activate it by type *M-x muse-mode RET*.

You will notice that Muse files are highlighted very simply. Links are colored blue, headings are large and bold text, and `<example>` tags are colored in grey.

There are several different ways to edit things like links, which hide the underlying Muse markup. One way is to toggle font-locking off by hitting *C-c C-l*, which is also *M-x font-lock-mode*, make changes, and then hit *C-c C-l* again to toggle font-locking back on. Another way is just to move into the text and edit it. Markup can also be removed by normal deletion methods, though some side effects might require a second deletion.

For the particular case of editing links, it is easiest to move to the link and do *C-c C-e*, which is also *M-x muse-edit-link-at-point*. This prompts you for the link and its description, using the previous contents of the link as initial values. A link to another Muse file may be created by hitting *C-c TAB l*. A link to a URL may be created by hitting *C-c TAB u*. Links may be followed by hitting *RET* on them.

If you want to add a new list item, this may by accomplished by hitting *M-RET*. This will put a dash and some spaces on the screen. The dash is the Muse markup that indicates a list item. It is also possible to created "nested" lists with this command, by adjusting the number of spaces in front of the dashes. If you have lists with long lines, you can move to a list item and hit *M-q* to wrap it onto multiple lines.

5.3 Publishing a Single File or Project

The command *M-x* `muse-project-publish-this-file` will publish the current document to any available publishing style (a publishing style is an output format, like HTML or Docbook), placing the output in the current directory. If you are in Muse Mode, this command will be bound to *C-c C-t*. If the file has been published recently, and its contents have not changed, running *C-c C-t* again will not publish the file. To force publishing in this case, do *C-u C-c C-t*.

If you have set up projects and are visiting a file that is part of a project, then *C-c C-t* will restrict the output formats to those which are used by the project, and will automatically publish to the output directory defined by the project. If you want to publish to a different directory or use a different format, then use *C-c M-C-t*, which is also *M-x* `muse-publish-this-file`.

If the currently opened file is part of a defined project in `muse-project-alist`, it (and the rest of the changed files in a project) may be published using *C-c C-p*.

5.4 Using a Different File Extension

By default, Muse expects all project files to have the file extension '`.muse`'. Files without this extension will not be associated with Muse mode and will not be considered part of any project, even if they are within a project directory.

If you don't want to use '`.muse`', you can customize the extension by setting the value of `muse-file-extension`.

If you don't want to use any extension at all, and want Muse to autodetect project files based on their location, then add the following to your Muse settings file.

```
(setq muse-file-extension nil
      muse-mode-auto-p t)
```

Note that if you chose to have `muse-file-extension` set to `nil`, you may have trouble if your '`.emacs`' file or other init scripts attempt to visit a Muse file. (A very common example of this is if you use Planner with Muse and run `(plan)` from your '`.emacs`'.) If you wish to visit Muse files from your '`.emacs`', be sure to also add the following additional code before any such visits happen:

```
(add-hook 'find-file-hooks 'muse-mode-maybe)
```

6 Creating and Managing Muse Projects

Often you will want to publish all the files within a directory to a particular set of output styles automatically. To support, Muse allows for the creation of "projects".

6.1 A Single-Project Example

Here is a sample project, which may be defined in your '.emacs' file.

```
(setq muse-project-alist
      '(("Website" ("~/Pages" :default "index")
         (:base "html" :path "~/public_html")
         (:base "pdf" :path "~/public_html/pdf"))))
```

The above defines a project named "website", whose files are located in the directory '~/Pages'. The default page to visit is 'index'. When this project is published, each page will be output as HTML to the directory '~/public_html', and as PDF to the directory '~/public_html/pdf'. Within any project page, you may create a link to other pages using the syntax '[[pagename]]'.

If you would like to include only some files from a directory in a Muse project, you may use a regexp in place of '~/Pages' in the example.

6.2 A Multiple-Project Example

It is possible to specify multiple projects. Here is an example of three projects: a generic website, a projects area, and a day-planner (the day-planner part requires Planner Mode—see http://wjsullivan.net/PlannerMode.html to get it).

```
(setq muse-project-alist
      '(("Website" ("~/Pages" :default "index")
         (:base "html" :path "~/public_html"))
        (("Projects" ("~/Projects" :default "index")
         (:base "xhtml"
                :path "~/public_html/projects"
                :exclude "/TopSecret")
         (:base "pdf"
                :path "~/public_html/projects/pdf"
                :exclude "/TopSecret")))
        ("Plans" ("~/Plans"
                  :default "TaskPool"
                  :major-mode planner-mode
                  :visit-link planner-visit-link)
         (:base "planner-xhtml"
                :path "~/public_html/plans"))))
```

The ':major-mode' attribute specifies which major to use when visiting files in this directory.

The ':visit-link' attribute specifies the function to call when visiting links.

The ':exclude' attribute has a regexp that matches files to never publish.

6.3 Publishing Subdirectories in Projects

If you want to publish a directory and all of its subdirectories, Muse provides two convenience functions that together generate the proper rules for you. Note that we use the backtick to begin this muse-project-alist definition, rather than a single quote.

```
(setq muse-project-alist
      `(("Website" ("~/Pages" :default "index")
         (:base "html" :path "~/public_html"))
        ("Blog" (,@(muse-project-alist-dirs "~/Blog")
                 :default "index")
         ;; Publish this directory and its subdirectories.  Arguments
         ;; are as follows.  The above `muse-project-alist-dirs' part
         ;; is also needed.
         ;;   1. Source directory
         ;;   2. Output directory
         ;;   3. Publishing style
         ;;   remainder: Other things to put in every generated style
         ,@(muse-project-alist-styles "~/Blog"
                                      "~/public_html/blog"
                                      "blosxom"))))
```

The `muse-project-alist-dirs` function takes a directory and returns it and all of its subdirectories in a list.

The `muse-project-alist-styles` function is explained by the comments above.

The "blosxom" text is the name of another publishing style, much like "html". See Section 9.1 [Blosxom], page 27, for further information about it. You can use any publishing style you like for the third argument to `muse-project-alist-styles`.

6.4 Listing of Available Options for Projects

This is a listing of all of the various options (or, more accurately: attributes) that may be specified in `muse-project-alist`.

Each muse-project-alist entry looks like this:

```
(PROJECT-NAME (SOURCES)
  OUTPUTS)
```

We refer to these names below.

"Attributes", which compose SOURCES and OUTPUTS, are a pair of values. The first value is a keyword, like '`:default`'. The second part is the value associated with that keyword, such as the text "index". If you are familiar with Emacs Lisp property lists, the concept is similar to that, except that in the SOURCES section, single directories can be interspersed with two-value attributes.

Project Name

This is a string that indicates the name of the project. It is primarily used for publishing interwiki links with the '`muse-wiki.el`' module.

Sources

This part of a muse-project-alist entry consists of two-value attributes, and also directory names. If you are publishing a book, the order of directories and attributes is significant.

The minimal content for the sources section is a list of directories.

'`:book-chapter`'
: Indicates a new chapter of a book. The text of the title of the chapter comes immediately after this keyword.

'`:book-end`'
: Indicates the end of a book. Directories listed after this one are ignored when publishing a book. The value "t" (without quotes) should come immediately after this keyword.

'`:book-funcall`'
: A function to call while publishing a book. This is useful for doing something just after a particular chapter.

'`:book-part`'
: Indicates the beginning of a new part of the book. The text of the title should come immediately after this keyword.

'`:book-style`'
: Indicate a particular publishing style to use for this part of the book. If this is specified, it should come just after a '`:part`' attribute.

'`:default`'
: The default page to visit when browsing a project. Also, if you are using the '`muse-wiki.el`' module, publishing a link to just a project's name will cause it to link to this default file.

'`:force-publish`'
: This specifies a list of pages which should be published every time a project is published (by using *C-c C-p*, for example), regardless of whether their contents have changed. This is useful for updating Index pages, pages that use the `<include>` tag, and other pages that have dynamically-generated content.

'`:major-mode`'
: This specifies the major mode to use when visiting files in this project. The default is `muse-mode`.

'`:nochapters`'
: This indicates that while publishing a book, do not automatically create chapters. Values which may follow this are nil (the default, which means that we automatically create chapters), or non-nil, which means that we manually specify chapters with the '`:book-chapter`' attribute,

'`:publish-project`'
: Indicates which function we should call when publishing a project.

'`:set`'
: This specifies a list of variables and values to set when publishing a project. The list should be a property list, which is in the form:

```
(VAR1 VALUE1 VAR2 VALUE2 ...)
```

'`:visit-link`'
: Specifies the function to call when visiting a link. The default is `muse-visit-link-default`. The arguments for that function should be (1) the link and (2) whether to visit the link in a new window.

Outputs

This part of a muse-project-alist entry is composed of lists of attributes. Each list is called an "output style".

The minimal content for an output style is a '`:base`' attribute and a '`:path`' attribute.

'`:base`'
: Publishing style to use, such as "html", "docbook", or "pdf".

'`:base-url`'
: An external URL which can be used to access published files. This is mainly used by the '`muse-wiki`' module when publishing links between two separate projects, if the projects are served on different domains.

  It is also used by the '`muse-journal`' module to create the RSS or RDF output.

'`:exclude`'
: Exclude items matching a regexp from being published. The regexp should usually begin with `"/"`.

'`:include`'
: Only include items matching a regexp when publishing. The regexp should usually begin with `"/"`.

'`:path`'
: The directory in which to store published files.

'`:timestamps`'
: A file containing the timestamps (that is, time of creation) for files in this project. It might eventually used by the '`muse-blosxom`' module, but this option is not currently in use by any Muse code.

7 Keys Used in Muse Mode

This is a summary of keystrokes available in every Muse buffer.

C-c C-a (‘muse-index’)
: Display an index of all known Muse pages.

C-c C-b (‘muse-find-backlinks’)
: Find all pages that link to this page.

C-c C-e (‘muse-edit-link-at-point’)
: Edit link at point.

C-c C-f (‘muse-project-find-file’)
: Open another Muse page. Prompt for the name.

C-c C-i l, C-c TAB l (‘muse-insert-relative-link-to-file’)
: Insert a link to a file interactively.

C-c C-i t, C-c TAB t (‘muse-insert-tag’)
: Insert a tag interactively.

C-c C-i u, C-c TAB u (‘muse-insert-url’)
: Insert a URL interactively.

C-c C-l (‘font-lock-mode’)
: Toggle font lock / highlighting for the current buffer.

C-c C-p (‘muse-project-publish’)
: Publish any Muse pages that have changed.

C-c C-s (‘muse-search’)
: Find text in all files of the current project.

C-c C-t (‘muse-project-publish-this-file’)
: Publish the currently-visited file. Prompt for the style if the current file can be published using more than one style.

C-c C-S-t, or C-c C-M-t (‘muse-publish-this-file’)
: Publish the currently-visited file. Prompt for both the style and output directory.

C-c C-v (‘muse-browse-result’)
: Show the published result of this page.

C-c = (‘muse-what-changed’)
: Diff this page against the last backup version.

TAB
: Move to the next Wiki reference.

S-TAB
: Move to the previous Wiki reference.

M-TAB
: Complete the name of a page from the current project at point.

M-RET
: Insert a new list item at point, indenting properly.

C-<
: Decrease the indentation of the list item at point.

C-> Increase the indentation of the list item at point.

M-x muse-colors-toggle-inline-images RET
: Toggle display of inlined images on/off.

M-x muse-update-values RET
: Update various values that are automatically generated.

: Call this after changing `muse-project-alist`.

8 Rules for Using Markup

A Muse document uses special, contextual markup rules to determine how to format the output result. For example, if a paragraph is indented, Muse assumes it should be quoted.

There are not too many markup rules, and all of them strive to be as simple as possible so that you can focus on document creation, rather than formatting.

8.1 Paragraphs: centering and quoting

Paragraphs in Muse must be separated by a blank line.

Centered paragraphs and quotations

A line that begins with six or more columns of whitespace (either tabs or spaces) indicates a centered paragraph. Alternatively, you can use the `<center>` tag to surround regions that are to be published as centered paragraphs.

But if a line begins with whitespace, though less than six columns, it indicates a quoted paragraph. Alternatively, you can use the `<quote>` tag to surround regions that are to be published as quoted paragraphs.

Literal paragraphs

The `<example>` tag is used for examples, where whitespace should be preserved, the text rendered in monospace, and any characters special to the output style escaped.

There is also the `<literal>` tag, which causes a marked block to be entirely left alone. This can be used for inserting a hand-coded HTML blocks into HTML output, for example.

If you want some text to only be inserted when publishing to a particular publishing style, use the '`style`' attribute for the `<literal>` tag. An example follows.

```
<literal style="latex">
A LaTeX-based style was used in the publishing of this document.
</literal>
```

This will leave the region alone if the current publishing style is "latex" or based on "latex", such as "pdf", and delete the region otherwise. It is also possible to leave the text alone only for one particular style, rather than its derivations, by adding `exact="t"` to the tag.

Line breaks

If you need a line break, then use the '`<br>`' tag. Most of the time this tag is unnecessary, because Muse will automatically detect paragraphs by means of blank lines. If you want to preserve newlines in several lines of text, then use verse markup instead (see Section 8.6 [Verse], page 17).

8.2 Levels of headings

A heading becomes a chapter or section in printed output – depending on the style. To indicate a heading, start a new paragraph with one or more asterices, followed by a space and the heading title. Then begin another paragraph to enter the text for that section.

All levels of headings will be published. Most publishing styles only distinguish the between the first 4 levels, however.

```
* First level

** Second level

*** Third level

**** Fourth level
```

8.3 Directives at the beginning of a document

Directives are lines beginning with the '#' character that come before any paragraphs or sections in the document. Directives are of the form "#directive content of directive". You can use any combination of uppercase and lowercase letters for directives, even if the directive is not in the list below.

The `muse-publishing-directive` function may be used in header and footer text to access directives. For example, to access the `#title` directive, use `(muse-publishing-directive "title")`.

The following is a list of directives that Muse uses.

`#author` The author of this document.

If this is not specified, Muse will attempt to figure it out from the `user-full-name` variable.

`#date` The date that the document was last modified.

This is used by publishing styles that are able to embed the date information.

`#desc` A short description of this document.

This is used by the `journal` publishing style to embed information inside of an RSS/RDF feed.

`#title` The title of this document.

If this is not specified, the name of the file is used.

8.4 Bold, italicized, and underlined text

To emphasize text, surround it with certain specially recognized characters.

```
*emphasis*
**strong emphasis**
***very strong emphasis***
_underlined_
=verbatim and monospace=
```

While editing a Muse document in Muse mode, these forms of emphasis will be highlighted in a WYSIWYG manner. Each of these forms may span multiple lines.

Verbatim text will be colored as gray by default. To change this, customize `muse-verbatim-face`.

You can also use the `<code>` tag to indicate verbatim and monospace text. This is handy for regions that have an "=" in them.

8.5 Making notes to be shown at the end

A footnote reference is simply a number in square brackets. To define the footnote, place this definition at the bottom of your file. '`footnote-mode`' can be used to greatly facilitate the creation of these kinds of footnotes.

Footnotes are defined by the same number in brackets occurring at the beginning of a line. Use footnote-mode's *C-c ! a* command, to very easily insert footnotes while typing. Use *C-x C-x* to return to the point of insertion.

8.6 Indicating poetic stanzas

Poetry requires that whitespace be preserved, but without resorting to monospace. To indicate this, use the following markup, reminiscent of email quotations.

```
> A line of Emacs verse;
>   forgive its being so terse.
```

You can also use the `<verse>` tag, if you prefer.

```
<verse>
A line of Emacs verse;
  forgive its being so terse.
</verse>
```

Multiple stanzas may be included in one set of `<verse>` tags, as follows.

```
<verse>
A line of Emacs verse;
  forgive its being so terse.

In terms of terse verse,
  you could do worse.
</verse>
```

8.7 Lists of items

Lists are given using special characters at the beginning of a line. Whitespace must occur before bullets or numbered items, to distinguish from the possibility of those characters occurring in a real sentence.

These are rendered as a bullet list.

```
Normal text.

 - bullet item one
 - bullet item two
```

An enumerated list follows.

```
Normal text.

 1. Enum item one
 2. Enum item two
```

Here is a definition list.

```
Term1 ::
  This is a first definition
  And it has two lines;
  no, make that three.

Term2 :: This is a second definition
```

Nested lists

It is possible to nest lists of the same or different kinds. The "level" of the list is determined by the amount of initial whitespace.

```
Normal text.

 - Level 1, bullet item one
   1. Level 2, enum item one
   2. Level 2, enum item two
 - Level 1, bullet item two
   1. Level 2, enum item three
   2. Level 2, enum item four
      term :: definition
```

Breaking list items

If you want to break up a line within any list type, just put one blank line between the end of the previous line and the beginning of the next line, using the same amount of initial indentation.

```
 - bullet item 1, line 1

   bullet item 1, line 2

   1. Enum line 1

      Enum line 2

 - bullet item 2, line 1

   bullet item 2, line 2
```

8.8 Generation of data tables

Only very simple tables are supported. The syntax is as follows.

```
Double bars  || Separate header fields

Single bars   | Separate body fields
Here are more | body fields

Triple bars ||| Separate footer fields
```

Some publishing styles require header fields to come first, then footer fields, and then the body fields. You can use any order for these sections that you like, and Muse will re-order them for you at publish-time.

If you wish to disable table generation for one Muse file, add the directive '`#disable-tables t`' to the top of the file.

Other table formats

It is possible to publish very basic Orgtbl-mode style tables.

```
| org  | style | table |
|------+-------+-------|
| one  |       | one   |
| two  | two   |       |
|      | three | three |
|------+-------+-------|
| more | stuff |       |
```

If you are used to the way that Org Mode publishes these tables, then customize 'muse-html-table-attributes' to the following, in order to get a similar kind of output.

```
border="2" cellspacing="0" cellpadding="6" rules="groups" frame="hsides"
```

'`table.el`' style tables are also supported, as long as '`table.el`' itself supports outputting tables for a particular publishing style. At the time of this writing, the "html", "latex", and "docbook" styles are supported by '`table.el`'. Styles derived from these styles will also work.

```
+---+-----+---+
|   | one | 1 |
+---+-----+---+
| b | two |   |
+---+-----+---+
| c |     | 3 |
+---+-----+---+
```

8.9 Hyperlinks and email addresses with descriptions

A hyperlink can reference a URL, or another page within a Muse project. In addition, descriptive text can be specified, which should be displayed rather than the link text in output styles that supports link descriptions. The syntax is as follows.

```
[[link target][link description]]
[[link target without description]]
```

Thus, the current maintainer's homepage for Muse can be found '`[[http://mwolson.org/projects/Ema`
or at '`[[http://mwolson.org/projects/EmacsMuse.html]]`'.

8.10 Bare URLs, WikiNames, and InterWiki links

A URL or email address encountered in the input text is published as a hyperlink. These kind of links are called *implicit links* because they are not separated from the rest of the Muse document in any way.

Some characters in URLs will prevent Muse from recognizing them as implicit links. If you want to link to a URL containing spaces or any of the characters “][,"’‘()<>^”, you will have to make the link explicit. The punctuation characters “.,;:” are also not recognized as part of a URL when they appear at its end. For information on how to make an explicit link, see Section 8.9 [Hyperlinks and email addresses with descriptions], page 19.

If the `muse-wiki` module is loaded, another form of implicit link will be made available. WikiNames, which are typed in CamelCase, are highlighted and published as links, provided that the file they refer to exists.

Customization of WikiName recognition may be accomplished by editing the `muse-wiki-wikiword-regexp` option and subsequently running `(muse-configure-highlighting 'muse-colors-markupmuse-colors-markup)`. If you use the Customize interface, the latter will be done automatically.

The `muse-wiki` module also allows for InterWiki links. These are similar to WikiWords, but they specify both the project and page of a file. The names of your project entries in `muse-project-alist` will be used as InterWiki names by default. Several examples follow.

```
Blog::DocumentingMuse
Projects#EmacsMuse
Website
```

In the first case, the interwiki delimiter is ‘`::`’, ‘`Blog`’ is the project name, and ‘`DocumentingMuse`’ is the page name. In the second example, ‘`#`’ is the interwiki delimiter. If the name of a project occurs by itself in text, like the third case, it will be colorized and published as a link to the default page of the given project.

Customization of interwiki links may be accomplished by editing the `muse-wiki-interwiki-alist` option.

It is also possible to link to an anchor in an interwiki document. This is called a “three-part link”. Examples of this follow.

```
Blog::DocumentingMuse#anchor1
Projects#EmacsMuse#anchor2
```

8.11 Publishing and displaying images

Image links

Links to images may be used in either the target or the description, or both. Thus, the following code will publish as a clickable image that points to `http://mwolson.org/`.

```
[[http://mwolson.org/][/static/logos/site-logo.png]]
```

Normally, images in the link part will be inlined.

If you want these images to be published as links instead, place the text “URL:” immediately in front of the link text. An example follows.

```
[[URL:http://mwolson.org/static/logos/site-logo.png]]
```

Displaying images in Muse mode

If a link to a locally-available image is encountered in the link description, Muse mode will attempt to display it if your version of Emacs permits this.

This behavior may be toggled with *C-c C-i*, or disabled permanently by setting the `muse-colors-inline-images` option to `nil`.

The method for finding images may be altered by customizing the `muse-colors-inline-image-method` option. One useful value for this option is `muse-colors-use-publishing-directory`, which tells Muse mode to look in the directory where the current file will be published. The default is to look in the current directory. Relative paths like '`../pics/`' should work for either setting.

Eventually, it is hoped that Muse will be able to copy images from the a "source" directory to a publishing directory by customizing `muse-project-alist`, but this has not been implemented yet.

Publishing simple images

The following example will display correctly and publish correctly if a PNG file called '`TestLogo.png`' exists in the '`../pics/`' directory. If text is on the same line as the picture, it will remain so in the output.

```
[[../myimage.png]]
```

Publishing images with captions

If you want to add a caption to an image, use the following syntax. This will center the image (if the output format supports it) and add a centered caption below the picture. Formats that do not support centering the image will instead leave it against the left margin.

```
[[../pics/mycat.png][My cat Dexter]]
```

Images with captions may only occur in their own paragraphs, with no text on the same line. Otherwise, the published output will not be syntactically correct.

8.12 Inserting a horizontal line or anchor

Horizontal Rules

Four or more dashes indicate a horizontal rule. Be sure to put blank lines around it, or it will be considered part of the proceeding or following paragraph!

Anchors

If you begin a line with "#anchor" – where "anchor" can be any word that doesn't contain whitespace – it defines an anchor at that point into the document. This point can be referenced using "page#anchor" as the target in a Muse link.

8.13 Evaluating Emacs Lisp code in documents for extensibility

Arbitrary kinds of markup can be achieved using the `<lisp>` tag. With the `<lisp>` tag, you may generate whatever output text you wish. The inserted output will get marked up if the `<lisp>` tag appears within the main text of the document.

```
<lisp>(concat "This form gets " "inserted")</lisp>
```

Note that you should not use the `insert` command within a set of `<lisp>` tags, since the return value from the `<lisp>` tags will be automatically inserted into the document.

It is also possible to treat the output as if it were surrounded by the `<example>`, `<src>`, or `<verse>` tags, by specifying "example", "src", or "verse" as the 'markup' attribute of the `<lisp>` tag.

```
<lisp markup="example">
(concat "Insert" " me")
</lisp>
```

Other languages also have tags that cause source code to be evaluated. See Section 8.16 [Tag Summary], page 23, for details.

8.14 Support for citing other resources

Example

Here is an example of what citations look like in a Muse document.

```
#bibsource REFDB

* Title
** Subtitle

Some text before <cite>Miller1999</cite> and after the citation.

This is an author-only citation <cite type="author">Miller1999</cite>.

And this is a year-only citation <cite type="year">Miller1999</cite>.

Finally, this is a multi-head citation
<cite>Miller1999,Andrews2005</cite>.
```

Overview

The `#bibsource` directive defines the source of the bibliographies. The following sources are possible.

- DocBook + RefDB: the string "REFDB"
- LaTeX + bibtex: the name of an appropriate bibtex file
- LaTeX + RefDB: if the input file is called "foo.muse", then set this to "foo.bib"

Citations are encoded as `<cite>` elements which enclose the citation keys as they are defined in the bibliography file or database. In multi-head citations, the citation keys have to be separated by colons or semicolons. The `latex` and `docbook` styles translate these to the proper separator automatically.

The `<cite>` elements take an optional "type" attribute that defines how the citation is rendered. If the attribute is missing, you'll get a regular citation according to the bibliography style, e.g." (Miller et al., 1999)". If the attribute is set to "author", only the name of the author(s) will be rendered. Accordingly, "year" will cause the year to be printed. This is useful to create citations like this:

```
Miller et al. had already shown in a previous publication (1999) that
this is not going to work.
```

Remember that refdb-mode (the Emacs interface to RefDB) can retrieve references by simply marking the citation key and running the `refdb-getref-by-field-on-region` command. Later versions of `refdb-mode` will also allow to insert references as Muse citations (which is already implemented for DocBook, TEI, and LaTeX documents).

You may have noticed that there is no element to indicate the position of the bibliography. The latter is always created at a valid position close to the end of the document. The functions `muse-docbook-bibliography` and `muse-latex-bibliography` are called in the header or footer to generate this content, so it is possible to change the exact position.

8.15 Lines to omit from published output

Use the following syntax to indicate a comment. Comments will not be published.

```
; Comment text goes here.
```

That is, only a semi-colon at the beginning of a line, followed by a literal space, will cause that line to be treated as a comment.

You can alternatively surround the region with the `<comment>` tag.

If you wish the comment to be published, but just commented out using the comment syntax of the output format, then set '`muse-publish-comments-p`' to non-nil.

8.16 Tags that Muse recognizes

Muse has several built-in tags that may prove useful during publishing. See [muse-publish-markup-tags], page 54, to see how to customize the tags that Muse uses, as well as make your own tags.

Only a small subset of these tags are available in header and footer text. The `muse-publish-markup-header-footer-tags` option lists the tags that are allowed in headers and footers.

Syntax

If a tag takes arguments, it will look like this, where "tagname" is the name of the tag.

```
<tagname arg1="string1" arg2="string2">
```

If you want the tag to look like it came straight from an XHTML document, you can alternatively do the following.

```
<tagname arg1="string1" arg2="string2" />
```

If a tag surrounds some text, it will look like this.

```
<tagname>Some text</tagname>
```

If a tag surrounds a large region, it will look like this.

```
<tagname>
Some text.
Some more text.
</tagname>
```

Tag listing

This is the complete list of tags that Muse accepts, including those that were mentioned in previous sections.

'
'
: Insert a line break.

  Muse will automatically detect paragraphs when publishing by means of blank lines, so this tag is usually unnecessary.

'<cite>'
: Insert a citation to another source.

  This takes the argument 'type', which indicates the type of citation. The valid types are "author" and "year". If this argument is omitted, include both author and year in the citation.

  The bibliography to use for the citation may be specified by the '#bibsource' directive.

  See Section 8.14 [Citations], page 22, for additional information.

'<class>'
: If publishing to HTML, surround the given text with a <span> tag. It takes one argument called "name" that specifies the "class" attribute of the <span> tag.

  If publishing to a different format, do nothing extra to the text.

'<code>'
: Treat the text surrounded by the tag as if they were enclosed in equal signs, that is, make it monospace.

'<command>'
: Run a command on the region, replacing the region with the result of the command. The command is specified with the "interp" argument. If no value for "interp" is given, pass the entire region to the shell.

  The "markup" argument controls how this section is marked up.

  If it is omitted, publish the region with the normal Muse rules.

  If "nil", do not mark up the region at all, but prevent Muse from further interpreting it.

  If "example", treat the region as if it was surrounded by the <example> tag.

  If "src", treat the included text as if it was surrounded by the <src> tag. You should also specify the "lang" attribute if doing this.

  If "verse", treat the region as if it was surrounded by the <verse> tag, to preserve newlines.

  Otherwise, it should be the name of a function to call, with the buffer narrowed to the region.

'<comment>'
: Treat the entire region as a comment. If the option *muse-publish-comments-p* is nil, delete the region, otherwise publish it using the comment syntax of the current publishing style.

'<contents>'
: Publish a Table of Contents. This will either be inserted in-place or at the beginning of the document, depending on your publishing style. It does not have a delimiting tag.

By default, only 2 levels of headings will be included in the generated Table of Contents. To change this globally, customize the *muse-publish-contents-depth* option. To change this only for the current tag, use the "depth" argument.

'`<div>`' Insert a <div> tag into HTML documents, and do not insert anything special for other non-HTML publishing formats.

If the "style" argument is provided, include it with the published `<div>` tag. Likewise for the "id" argument.

'`<example>`'
Publish the region in monospace, preserving the newlines in the region. This is useful for snippets of code.

'`<include>`'
Insert the given file at the current location during publishing. The basic use of this tag is as follows, replacing "included_file" with the name of the file that you want to include.

```
<include file="included_file">
```

The "markup" argument controls how this section is marked up.

If it is omitted, publish the included text with the normal Muse rules.

If "nil", do not mark up the included text at all.

If "example", treat the included text as if it was surrounded by the `<example>` tag.

If "src", treat the included text as if it was surrounded by the `<src>` tag. You should also specify the "lang" attribute if doing this.

If "verse", treat the included text as if it was surrounded by the `<verse>` tag, to preserve newlines.

Otherwise, it should be the name of a function to call after inserting the file with the buffer narrowed to the section inserted.

'`<lisp>`' Evaluate the Emacs Lisp expressions between the initial and ending tags. The result is then inserted into the document, so you do not need to explicitly call `insert`. All text properties are removed from the resulting text.

This tag takes the "markup" argument. See the description of `<command>` for details.

'`<literal>`'
Make sure that the text enclosed by this tag is published without escaping it in any way. This is useful for inserting markup directly into the published document, when Muse does not provide the desired functionality.

'`<markup>`'
Mark up the text between the initial and ending tags. The markup command to use may be specified by the "function" argument. The standard Muse markup routines are used by default if no "function" argument is provided.

This is useful for marking up regions in headers and footers. One example that comes to mind is generating a published index of all of the files in the current project by doing the following.

```
<markup><lisp>(muse-index-as-string t t)</lisp></markup>
```

'<perl>' Run the `perl` language interpreter on the region, replacing the region with the result of the command.

This tag takes the "markup" argument. See the description of `<command>` for details.

'<python>'
Run the `python` language interpreter on the region, replacing the region with the result of the command.

This tag takes the "markup" argument. See the description of `<command>` for details.

'<quote>' Publish the region as a blockquote. This will either be inserted in-place or at the beginning of the document, depending on your publishing style. It does not have a delimiting tag.

'<ruby>' Run the `ruby` language interpreter on the region, replacing the region with the result of the command.

This tag takes the "markup" argument. See the description of `<command>` for details.

'<src>' Publish the region using htmlize. The language to use may be specified by the "lang" attribute.

Muse will look for a function named *lang*-mode, where *lang* is the value of the "lang" attribute.

This tag requires htmlize 1.34 or later in order to work. If this is not satisfied, or the current publishing style is not HTML-based, Muse will publish the region like an `<example>` tag.

'<verbatim>'
This is used when you want to prevent Muse from trying to interpret some markup. Surround the markup in `<verbatim>` and `</verbatim>`, and it will not be interpreted.

This tag was used often in previous versions of Muse because they did not support whole-document escaping of specials. Now, it will only be needed for other tags, and perhaps footnotes as well.

'<verse>' Preserve the newlines in the region. In formats like HTML, newlines are removed by default, hence the need for this tag. In other publishing styles, this tag may cause the text to be indented slightly in a way that looks nice for poetry and prose.

9 Publishing Various Types of Documents

One of the principle features of Muse is the ability to publish a simple input text to a variety of different output styles. Muse also makes it easy to create new styles, or derive from an existing style.

9.1 Integrating Muse and pyblosxom.cgi

The Blosxom publishing style publishes a tree of categorised files to a mirrored tree of stories to be served by blosxom.cgi or pyblosxom.cgi. In other words, each blog entry corresponds with one file.

9.1.1 Other tools needed for the Blosxom style

You will need to have `pyblosxom.cgi` or `blosxom.cgi` installed on a machine that you have upload access to.

The major difficulty in both of these programs is specifying the date of the entries. Both programs rely on the file modification time rather than any data contained in the entries themselves. A plugin is needed in order for these programs to be able to get the correct date.

PyBlosxom

There are two different ways of accomplishing this in pyblosxom. The first way involves gathering the timestamps (as specified by the `#date` directive) into one file and then sending that file along with published entries to the webserver.

The second will read each file at render time and parse the `#postdate` directive. Muse will translate the `#date` directive into `#postdate` at publish time, so you don't have to do any extra work.

Placing timestamps in one file

The following additional components are required in order to make the date of blog entries display as something sensible.

1. A script to gather date directives from the entire blog tree into a single file. The file must associate a blog entry with a date.
2. A plugin for (py)blosxom that reads this file.

These 2 things are provided for `pyblosxom.cgi` in the '`contrib/pyblosxom`' subdirectory. '`getstamps.py`' provides the former service, while '`hardcodedates.py`' provides the latter service.

Here is a sample listing from my '`timestamps`' file, which maps each file to a date. This can really be in any format, as long as your date-gathering script and your plugin can both understand it.

```
2005-04-01-14-16 personal/paper_cranes
2005-03-21 personal/spring_break_over
2004-10-24 personal/finished_free_culture
```

The script '`contrib/pyblosxom/make-blog`' demonstrates how to call '`getstamps.py`'. Note that you will need to set the current directory to where your Muse files are, execute '`getstamps.py`', and then move the generated timestamps file to your publishing directory.

Getting timestamp from entry while rendering

Alternately, the pyblosxom metadate plugin may be used. On the plus side, there is no need to run a script to gather the date. On the downside, each entry is read twice rather than once when the page is rendered. Set the value of `muse-blosxom-use-metadate` to non-nil to enable adding a `#postdate` directive to all published files. You can do this by:

```
M-x customize-variable RET muse-blosxom-use-metadate RET
```

With the metadate plugin installed in pyblosxom, the date set in this directive will be used instead of the file's modification time. The plugin is included with Muse at '`contrib/pyblosxom/metadate.py`'.

Blosxom

It is also possible to use Blosxom, which is written in Perl, to serve blog entries that were published with Muse. The steps are as follows.

1. Download and install blosxom from `http://blosxom.sourceforge.net/`.
2. Install the metadate plugin. It is available in '`contrib/blosxom/metadate_0_0_3`'.
3. Every time you make a new blog entry, change to the blosxom data directory and execute the '`contrib/blosxom/getstamps.pl`' script. This script has only recently been made, and may still have some bugs, so use with caution.

9.1.2 Format of a Blosxom entry and automation

Each Blosxom file must include '#date yyyy-mm-dd', or optionally the longer '#date yyyy-mm-dd-hh-mm', a title (using the `#title` directive), plus whatever normal content is desired.

The date directive is not used directly by `pyblosxom.cgi` or this program. You need to have the two additional items from the former section to make use of this feature.

There is a function called `muse-blosxom-new-entry` that will automate the process of making a new blog entry. To make use of it, do the following.

- Customize `muse-blosxom-base-directory` to the location that your blog entries are stored.
- Assign the `muse-blosxom-new-entry` function to a key sequence. I use the following code to assign this function to *`C-c p l'`*.

  ```
  (global-set-key "\C-cpl" 'muse-blosxom-new-entry)
  ```
- You should create your directory structure ahead of time under your base directory. These directories, which correspond with category names, may be nested.
- When you enter this key sequence, you will be prompted for the category of your entry and its title. Upon entering this information, a new file will be created that corresponds with the title, but in lowercase letters and having special characters converted to underscores. The title and date directives will be inserted automatically.

9.1.3 Blosxom styles and options provided

The following styles and options are available in the Blosxom publishing style.

Styles provided

`blosxom-html`
: Publish Blosxom entries in HTML form.

`blosxom-xhtml`
: Publish Blosxom entries in XHTML form.

Options provided

`muse-blosxom-extension`
: Default file extension for publishing Blosxom files.

`muse-blosxom-header`
: Header used for publishing Blosxom files.

  This may be text or a filename.

`muse-blosxom-footer`
: Footer used for publishing Blosxom files.

  This may be text or a filename.

`muse-blosxom-base-directory`
: Base directory of blog entries, used by `muse-blosxom-new-entry`.

  This is the top-level directory where your blog entries may be found locally.

9.2 Publishing entries into a compilation

This publishing style is used to output "books" in LaTeX or PDF format.

Each page will become a separate chapter in the book, unless the style keyword '`:nochapters`' is used, in which case they are all run together as if one giant chapter.

One way of publishing a book is to make a project for it, add the project to `muse-project-alist`, and use the `book-pdf` style with a very specific '`:include`' value to specify some page whose contents will be checked for the values of `#title` and `#date`, and whose name will be used in the output file. Then to publish the book, visit the aforementioned page and use *C-c C-t* or *C-c C-p* to trigger the publishing process. An example `muse-project-alist` for this method follows.

```
(setq muse-project-alist
      '(("MyNotes" (:nochapters t  ; do automatically add chapters
                    :book-chapter "Computer Science"
                    "~/Notes/cs"
                    :book-chapter "Mathematics"
                    "~/Notes/math"
                    :book-chapter "Emacs"
                    "~/Notes/emacs"
                    :book-end t ; the rest will not be placed in the book
                    "~/Notes"   ; so we can find the notes-anthology page
                    "~/Notes/private"
                    :force-publish ("index")
                    :default "index")
```

```
(:base "book-pdf"
       :include "/notes-anthology[^/]*$"
       :path "~/public_html/notes")
;; other publishing styles for each directory go here,
;; if desired
)))
```

In this example, there would be a file called '~/Notes/notes-anthology.muse', which would contain just the following. The resulting book would be published to '~/public_html/notes/notes-anthology.pdf'.

```
#title My Technology Ramblings
```

Another way is to call the `muse-book-publish-project` function manually, with a custom project entry. An example of this may be found in John Wiegley's configuration file at 'examples/johnw/muse-init.el', in the `muse-publish-my-books` function.

Styles provided

`book-latex`
: Publish a book in LaTeX form. The header and footer are different than the normal LaTeX publishing mode.

`book-pdf`
: Publish a book in PDF form. The header and footer are different than the normal PDF publishing mode.

Options provided

`muse-book-before-publish-hook`
: A hook run in the book buffer before it is marked up.

`muse-book-after-publish-hook`
: A hook run in the book buffer after it is marked up.

`muse-book-latex-header`
: Header used for publishing books to LaTeX.

  This may be text or a filename.

`muse-book-latex-footer`
: Footer used for publishing books to LaTeX.

  This may be text or a filename.

9.3 Publishing ConTeXt documents

This publishing style is capable of producing ConTeXt or PDF documents.

If you wish to publish PDF documents based on ConTeXt, you will need to have it installed. For Debian and Ubuntu, this can be accomplished by installing the "texlive" package.

Styles provided

`context`
: Publish a ConTeXt document.

`context-pdf`
: Publish a PDF document, using an external ConTeXt document conversion tool.

`context-slides`
: Produce slides from a ConTeXt document.

  Here is an example of a slide.

```
* First Slide

[[Some-sort-of-cute-image.png]]

** A subheading

 - A bullet point.
 - Another bullet point.

* Second Slide

... and so on
```

`context-slides-pdf`
: Publish a PDF document of ConTeXt slides.

Options provided

`muse-context-extension`
: Default file extension for publishing ConTeXt files.

`muse-context-pdf-extension`
: Default file extension for publishing ConTeXt files to PDF.

`muse-context-pdf-program`
: The program that is called to generate PDF content from ConTeXt content.

`muse-context-pdf-cruft`
: Extensions of files to remove after generating PDF output successfully.

`muse-context-header`
: Header used for publishing ConTeXt files.

  This may be text or a filename.

`muse-context-footer`
: Footer used for publishing ConTeXt files.

  This may be text or a filename.

`muse-context-markup-regexps`
: List of markup regexps for identifying regions in a Muse page.

  For more on the structure of this list, See [muse-publish-markup-regexps], page 47.

`muse-context-markup-functions`
: An alist of style types to custom functions for that kind of text.

For more on the structure of this list, See [muse-publish-markup-functions], page 47.

`muse-context-markup-strings`
: Strings used for marking up text.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-context-slides-header`
: Header for publishing a presentation (slides) using ConTeXt.

  Any of the predefined modules, which are available in the tex/context/base directory, can be used by writing a "module" directive at the top of the Muse file; if no such directive is provided, module pre-01 is used. Alternatively, you can use your own style ("mystyle", in this example) by replacing "\usemodule[]" with "\input mystyle".

  This may be text or a filename.

`muse-context-slides-markup-strings`
: Strings used for marking up text in ConTeXt slides.

`muse-context-markup-specials-document`
: A table of characters which must be represented specially. These are applied to the entire document, sans already-escaped regions.

`muse-context-markup-specials-example`
: A table of characters which must be represented specially. These are applied to `example>` regions.

  With the default interpretation of `<example>` regions, no specials need to be escaped.

`muse-context-markup-specials-literal`
: A table of characters which must be represented specially. This applies to =monospaced text= and `<code>` regions.

`muse-context-markup-specials-url`
: A table of characters which must be represented specially. These are applied to URLs.

`muse-context-markup-specials-image`
: A table of characters which must be represented specially. These are applied to image filenames.

`muse-context-permit-contents-tag`
: If nil, ignore `<contents>` tags. Otherwise, insert table of contents.

  Most of the time, it is best to have a table of contents on the first page, with a new page immediately following. To make this work with documents published in both HTML and ConTeXt, we need to ignore the `<contents>` tag.

  If you don't agree with this, then set this option to non-nil, and it will do what you expect.

9.4 Publishing in DocBook XML form

This publishing style is used to generate DocBook XML files.

Styles provided

`docbook` Publish a file in Docbook form.

Options provided

This publishing style uses the same options for markup up special characters as the "xml" publishing style. See Section 9.11 [XML], page 45, for details.

`muse-docbook-extension`
: Default file extension for publishing DocBook XML files.

`muse-docbook-header`
: Header used for publishing DocBook XML files.

  This may be text or a filename.

`muse-docbook-footer`
: Footer used for publishing DocBook XML files.

  This may be text or a filename.

`muse-docbook-markup-regexps`
: List of markup rules for publishing a Muse page to DocBook XML.

`muse-docbook-markup-functions`
: An alist of style types to custom functions for that kind of text.

`muse-docbook-markup-strings`
: Strings used for marking up text.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-docbook-encoding-default`
: The default Emacs buffer encoding to use in published files. This will be used if no special characters are found.

`muse-docbook-charset-default`
: The default DocBook XML charset to use if no translation is found in `muse-xml-encoding-map`.

9.5 Publishing in HTML or XHTML form

This publishing style is capable of producing HTML or XHTML documents.

Styles provided

`html` Supports publishing to HTML 4.0 and HTML 4.01, Strict or Transitional.

`xhtml` Supports publishing to XHTML 1.0 and XHTML 1.1, Strict or Transitional.

Options provided

If an HTML option does not have a corresponding XHTML option, it will be used for both of these publishing styles.

These publishing styles use the same options for markup up special characters as the "xml" publishing style. See Section 9.11 [XML], page 45, for details.

`muse-html-extension`
: Default file extension for publishing HTML files.

`muse-xhtml-extension`
: Default file extension for publishing XHTML files.

`muse-html-style-sheet`
: Store your stylesheet definitions here.

  This is used in `muse-html-header`. You can put raw CSS in here or a `<link>` tag to an external stylesheet. This text may contain `<lisp>` markup tags.

  If you are publishing to XHTML, then customize the `muse-xhtml-style-sheet` option instead.

`muse-xhtml-style-sheet`
: Store your stylesheet definitions here.

  This is used in `muse-xhtml-header`. You can put raw CSS in here or a `<link>` tag to an external stylesheet. This text may contain `<lisp>` markup tags.

`muse-html-header`
: Header used for publishing HTML files.

  This may be text or a filename.

`muse-html-footer`
: Footer used for publishing HTML files.

  This may be text or a filename.

`muse-xhtml-header`
: Header used for publishing XHTML files.

  This may be text or a filename.

`muse-xhtml-footer`
: Footer used for publishing XHTML files.

  This may be text or a filename.

`muse-html-anchor-on-word`
: When true, anchors surround the closest word.

  This allows you to select them in a browser (i.e. for pasting), but has the side-effect of marking up headers in multiple colors if your header style is different from your link style.

`muse-html-table-attributes`
: The attribute to be used with HTML `<table>` tags.

  If you want to make more-complicated tables in HTML, surround the HTML with the `literal` tag, so that it does not get escaped.

`muse-html-markup-regexps`
: List of markup rules for publishing a Muse page to HTML.

`muse-html-markup-functions`
: An alist of style types to custom functions for that kind of text.

`muse-html-markup-strings`
: Strings used for marking up text as HTML.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-xhtml-markup-strings`
: Strings used for marking up text as XHTML.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-html-markup-tags`
: A list of tag specifications, for specially marking up HTML. See [muse-publish-markup-tags], page 54, for more information.

`muse-html-meta-http-equiv`
: The http-equiv attribute used for the HTML `<meta>` tag.

`muse-html-meta-content-type`
: The content type used for the HTML `<meta>` tag.

  If you are striving for XHTML 1.1 compliance, you may want to change this to "application/xhtml+xml".

`muse-html-meta-content-encoding`
: The charset to append to the HTML `<meta>` tag.

  If set to the symbol 'detect, use `muse-xml-encoding-map` to try and determine the HTML charset from emacs's coding. If set to a string, this string will be used to force a particular charset.

`muse-html-charset-default`
: The default HTML meta charset to use if no translation is found in `muse-xml-encoding-map`.

`muse-html-encoding-default`
: The default Emacs buffer encoding to use in published files. This will be used if no special characters are found.

9.6 Integrating with ikiwiki

Note: Support for Ikiwiki is not yet complete. Use at your own risk.

Ikiwiki is a wiki compiler (`http://ikiwiki.info/`). Emacs Muse can (not yet) be used as a source format for Ikiwiki pages with the plugin '`IkiWiki::Plugin::muse`'.

The '`lisp/muse-ikiwiki.el`' file provides publishing functions and styles for Ikiwiki. The plugin for Ikiwiki to recognize Muse files is provided by the '`contrib/ikiwiki/IkiWiki/Plugin/muse.pm`' file. Two sample init files are available in the 'examples/ikiwiki' directory. Configure your 'ikiwiki.setup' file so that the muse_init variable has the location of your Muse init file.

If you are using CGI, The directory 'contrib/ikiwiki/IkiWiki' must be copied to the same directory as the CGI script that Ikiwiki generates. When publishing your wiki, the *PERL5LIB* environment variable must contain the path to the 'contrib/ikiwiki/IkiWiki' directory.

Styles provided

`ikiwiki` Supports publishing XHTML output that Ikiwiki can understand.

Options provided

`muse-ikiwiki-header`
: Header used for publishing Ikiwiki output files.

  This may be text or a filename.

`muse-ikiwiki-footer`
: Footer used for publishing Ikiwiki output files.

  This may be text or a filename.

Other relevant options

`muse-colors-evaluate-lisp-tags`
: Specify whether to evaluate the contents of `<lisp>` tags at display time. If nil, don't evaluate them. If non-nil, evaluate them.

  The actual contents of the buffer are not changed, only the displayed text.

`muse-html-src-allowed-modes`
: Modes that we allow the `<src>` tag to colorize. If `t`, permit the `<src>` tag to colorize any mode.

  If a list of mode names, such as `'("html" "latex")`, and the lang argument to `<src>` is not in the list, then use fundamental mode instead.

`muse-publish-enable-dangerous-tags`
: If non-nil, publish tags like `<lisp>` and `<command>` that can call external programs or expose sensitive information. Otherwise, ignore tags like this.

  This is useful to set to `nil` when the file to publish is coming from an untrusted source.

9.7 Keeping a journal or blog

The module facilitates the keeping and publication of a journal. When publishing to HTML, it assumes the form of a web log, or blog.

The input format for each entry is as follows.

```
* 20040317: Title of entry

text for the entry.
```

```
<qotd>
"You know who you are. It comes down to a simple gut check: You
either love what you do or you don't. Period." -- P. Bronson
</qotd>
```

The "qotd", or Quote of the Day, is entirely optional. When generated to HTML, this entry is rendered as the following.

```
<div class="entry">
  <div class="entry-qotd">
    <h3>Quote of the Day:</h3>
    <p>"You know who you are. It comes down to a simple gut
      check: You either love what you do or you don't. Period."
      -- P. Bronson</p>
  </div>
  <div class="entry-body">
    <div class="entry-head">
      <div class="entry-date">
        <span class="date">March 17, 2004</span>
      </div>
      <div class="entry-title">
        <h2>Title of entry</h2>
      </div>
    </div>
    <div class="entry-text">
      <p>Text for the entry.</p>
    </div>
  </div>
</div>
```

The plurality of "div" tags makes it possible to display the entries in any form you wish, using a CSS style.

Also, an .RDF file can be generated from your journal by publishing it with the "rdf" style. It uses the first two sentences of the first paragraph of each entry as its "description", and auto-generates tags for linking to the various entries.

muse-project-alist considerations

If you wish to publish an RDF or RSS feed, it is important to include the '`:base-url`' attribute in your `muse-project-alist` entry for your Journal projects. An example follows.

```
(setq muse-project-alist
      '(("Journal" ("~/Journal/"
                    :default "journal")
        (:base "journal-rss"
               :base-url "http://example.org/journal/"
               :path "~/public_html/journal"))))
```

Styles provided

`journal-html`
: Publish journal entries as an HTML document.

`journal-xhtml`
Publish journal entries as an XHTML document.

`journal-latex`
Publish journal entries as a LaTeX document.

`journal-pdf`
Publish journal entries as a PDF document.

`journal-book-latex`
Publish journal entries as a LaTeX book.

`journal-book-pdf`
Publish journal entries as a PDF book.

`journal-rdf`
Publish journal entries as an RDF file (RSS 1.0).

`journal-rss`
Publish journal entries as an RSS file (RSS 2.0).

`journal-rss-entry`
Used internally by `journal-rss` and `journal-rdf` for publishing individual entries.

Options provided

`muse-journal-heading-regexp`
A regexp that matches a journal heading.

Paren group 1 is the ISO date, group 2 is the optional category, and group 3 is the optional heading for the entry.

`muse-journal-date-format`
Date format to use for journal entries.

`muse-journal-html-heading-regexp`
A regexp that matches a journal heading from an HTML document.

Paren group 1 is the ISO date, group 2 is the optional category, and group 3 is the optional heading for the entry.

`muse-journal-html-entry-template`
Template used to publish individual journal entries as HTML.

This may be text or a filename.

`muse-journal-latex-section`
Template used to publish a LaTeX section.

`muse-journal-latex-subsection`
Template used to publish a LaTeX subsection.

`muse-journal-markup-tags`
A list of tag specifications, for specially marking up Journal entries.

See [muse-publish-markup-tags], page 54, for more information.

This is used by `journal-latex` and its related styles, as well as the `journal-rss-entry` style, which both `journal-rdf` and `journal-rss` use.

`muse-journal-rdf-extension`
: Default file extension for publishing RDF (RSS 1.0) files.

`muse-journal-rdf-base-url`
: The base URL of the website referenced by the RDF file.

`muse-journal-rdf-header`
: Header used for publishing RDF (RSS 1.0) files.

  This may be text or a filename.

`muse-journal-rdf-footer`
: Footer used for publishing RDF (RSS 1.0) files.

  This may be text or a filename.

`muse-journal-rdf-date-format`
: Date format to use for RDF entries.

`muse-journal-rdf-entry-template`
: Template used to publish individual journal entries as RDF.

  This may be text or a filename.

`muse-journal-rdf-summarize-entries`
: If non-nil, include only summaries in the RDF file, not the full data.

  The default is nil, because this annoys some subscribers.

`muse-journal-rss-heading-regexp`
: A regexp that matches a journal heading from an HTML document.

  Paren group 1 is the ISO date, group 2 is the optional category, and group 3 is the optional heading for the entry.

`muse-journal-rss-extension`
: Default file extension for publishing RSS 2.0 files.

`muse-journal-rss-base-url`
: The base URL of the website referenced by the RSS file.

`muse-journal-rss-header`
: Header used for publishing RSS 2.0 files.

  This may be text or a filename.

`muse-journal-rss-footer`
: Footer used for publishing RSS 2.0 files.

  This may be text or a filename.

`muse-journal-rss-date-format`
: Date format to use for RSS 2.0 entries.

`muse-journal-rss-entry-template`
: Template used to publish individual journal entries as RSS 2.0.

  This may be text or a filename.

`muse-journal-rss-enclosure-types-alist`
: File types that are accepted as RSS enclosures.

This is an alist that maps file extension to content type.

Useful for podcasting.

`muse-journal-rss-summarize-entries`
: If non-nil, include only summaries in the RSS file, not the full data.

  The default is nil, because this annoys some subscribers.

`muse-journal-rss-markup-regexps`
: List of markup rules for publishing a Muse journal page to RSS.

  For more information on the structure of this list, See [muse-publish-markup-regexps], page 47.

`muse-journal-rss-markup-functions`
: An alist of style types to custom functions for that kind of text.

  For more on the structure of this list, See [muse-publish-markup-functions], page 47.

9.8 Publishing LaTeX documents

This publishing style is capable of producing LaTeX or PDF documents.

If you wish to publish PDF documents, you will need to have a good LaTeX installation. For Debian and Ubuntu, this can be accomplished by installing the "tetex-bin" and "tetex-extra" packages. TeX fonts are also a must.

If your LaTeX installation has the file '`grffile.sty`', which may be found in the '`texlive-latex-recommended`' package for Debian and Ubuntu, then consider using it by adding the following to your header file. This allows spaces in filenames to work.

```
\usepackage{grffile}
```

Styles provided

`latex` Publish a LaTeX document.

`pdf` Publish a PDF document, using an external LaTeX document conversion tool.

`latexcjk` Publish a LaTeX document with CJK (Chinese) encodings.

`pdfcjk` Publish a PDF document with CJK (Chinese) encodings, using an external LaTeX document conversion tool.

`slides` Publish a LaTeX document that uses the Beamer extension. This is suitable for producing slides.

Here is an example of a slide.

```
<slide title="First Slide">
Everything between the slide tags composes this slide.

[[Some-sort-of-cute-image.png]]

 - A bullet point.
 - Another bullet point.
</slide>
```

`slides-pdf`
: Publish a PDF document of slides, using the Beamer extension.

`lecture-notes`
: Publish a LaTeX document that uses the Beamer extension. This is suitable for producing lecture notes.

  This can also use the `<slide>` tag.

`lecture-notes-pdf`
: Publish a PDF document of lecture notes, using the Beamer extension.

Options provided

`muse-latex-extension`
: Default file extension for publishing LaTeX files.

`muse-latex-pdf-extension`
: Default file extension for publishing LaTeX files to PDF.

`muse-latex-pdf-browser`
: The program to use when browsing a published PDF file.

  This should be a format string.

`muse-latex-pdf-program`
: The program that is called to generate PDF content from LaTeX content.

`muse-latex-pdf-cruft`
: Extensions of files to remove after generating PDF output successfully.

`muse-latex-header`
: Header used for publishing LaTeX files.

  This may be text or a filename.

`muse-latex-footer`
: Footer used for publishing LaTeX files.

  This may be text or a filename.

`muse-latexcjk-header`
: Header used for publishing LaTeX files (CJK).

  This may be text or a filename.

`muse-latexcjk-footer`
: Footer used for publishing LaTeX files (CJK).

  This may be text or a filename.

`muse-latex-slides-header`
: Header for publishing of slides using LaTeX.

  This may be text or a filename.

  You must have the Beamer extension for LaTeX installed for this to work.

`muse-latex-lecture-notes-header`
: Header publishing of lecture notes using LaTeX.

  This may be text or a filename.

  You must have the Beamer extension for LaTeX installed for this to work.

`muse-latex-markup-regexps`
: List of markup regexps for identifying regions in a Muse page.

  For more on the structure of this list, See [muse-publish-markup-regexps], page 47.

`muse-latex-markup-functions`
: An alist of style types to custom functions for that kind of text.

  For more on the structure of this list, See [muse-publish-markup-functions], page 47.

`muse-latex-markup-strings`
: Strings used for marking up text.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-latex-slides-markup-tags`
: A list of tag specifications, for specially marking up LaTeX slides.

`muse-latexcjk-encoding-map`
: An alist mapping emacs coding systems to appropriate CJK codings. Use the base name of the coding system (ie, without the -unix).

`muse-latexcjk-encoding-default`
: The default Emacs buffer encoding to use in published files.

  This will be used if no special characters are found.

`muse-latex-markup-specials-document`
: A table of characters which must be represented specially. These are applied to the entire document, sans already-escaped regions.

`muse-latex-markup-specials-example`
: A table of characters which must be represented specially. These are applied to `example>` regions.

  With the default interpretation of `<example>` regions, no specials need to be escaped.

`muse-latex-markup-specials-literal`
: A table of characters which must be represented specially. This applies to =monospaced text= and `<code>` regions.

`muse-latex-markup-specials-url`
: A table of characters which must be represented specially. These are applied to URLs.

`muse-latex-markup-specials-image`
: A table of characters which must be represented specially. These are applied to image filenames.

`muse-latex-permit-contents-tag`
: If nil, ignore `<contents>` tags. Otherwise, insert table of contents.

  Most of the time, it is best to have a table of contents on the first page, with a new page immediately following. To make this work with documents published in both HTML and LaTeX, we need to ignore the `<contents>` tag.

If you don't agree with this, then set this option to non-nil, and it will do what you expect.

9.9 Publish a poem to LaTeX or PDF

The `muse-poem` module makes it easy to attractively publish and reference poems in the following format, using the "memoir" module for LaTeX publishing. It will also markup poems for every other output style, though none are nearly as pretty.

```
Title


Body of poem


Annotations, history, notes, etc.
```

Once a poem is written in this format, just publish it to PDF using the `poem-pdf` style. To make an inlined reference to a poem that you've written – for example, from a blog page – there is a "poem" tag defined by this module.

```
<poem title="name.of.poem.page">
```

Let's assume the template above was called '`name.of.poem.page`'; then the above tag would result in this inclusion.

```
** Title

> Body of poem
```

John Wiegley uses this module for publishing all of the poems on his website, which are at `http://www.newartisans.com/johnw/poems.html`.

Styles provided

`poem-latex`
: Publish a poem in LaTeX form.

`poem-pdf`
: Publish a poem to a PDF document.

`chapbook-latex`
: Publish a book of poems in LaTeX form.

`chapbook-pdf`
: Publish a book of poems to a PDF document.

Options provided

`muse-poem-latex-header`
: Header used for publishing LaTeX poems.

  This may be text or a filename.

`muse-poem-latex-footer`
: Footer used for publishing LaTeX files.

  This may be text or a filename.

`muse-poem-markup-strings`
: Strings used for marking up poems.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-chapbook-latex-header`
: Header used for publishing a book of poems in LaTeX form.

  This may be text or a filename.

`muse-chapbook-latex-footer`
: Footer used for publishing a book of poems in LaTeX form.

  This may be text or a filename.

`muse-poem-chapbook-strings`
: Strings used for marking up books of poems.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

9.10 Publish entries to Texinfo format or PDF

Rules for publishing a Muse file as a Texinfo article.

Styles provided

`texi` Publish a file in Texinfo form.

`info` Generate an Info file from a Muse file.

`info-pdf` Publish a file in PDF form.

Options provided

`muse-texinfo-process-natively`
: If non-nil, use the Emacs 'texinfmt' module to make Info files.

`muse-texinfo-extension`
: Default file extension for publishing Texinfo files.

`muse-texinfo-info-extension`
: Default file extension for publishing Info files.

`muse-texinfo-pdf-extension`
: Default file extension for publishing PDF files.

`muse-texinfo-header`
: Text to prepend to a Muse page being published as Texinfo.

  This may be text or a filename. It may contain `<lisp>` markup tags.

`muse-texinfo-footer`
: Text to append to a Muse page being published as Texinfo.

  This may be text or a filename. It may contain `<lisp>` markup tags.

`muse-texinfo-markup-regexps`
: List of markup rules for publishing a Muse page to Texinfo.

  For more on the structure of this list, See [muse-publish-markup-regexps], page 47.

`muse-texinfo-markup-functions`
: An alist of style types to custom functions for that kind of text.

  For more on the structure of this list, See [muse-publish-markup-functions], page 47.

`muse-texinfo-markup-strings`
: Strings used for marking up text.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-texinfo-markup-specials`
: A table of characters which must be represented specially.

`muse-texinfo-markup-specials`
: A table of characters which must be represented specially. These are applied to URLs.

9.11 Publish entries to XML

Muse is capable of publishing XML documents, with the help of the '`muse-xml.el`' module.

A RelaxNG schema is available as part of the Muse distribution in the '`etc/muse.rnc`' file.

Styles provided

`xml` Publish a file in XML form.

Options provided

`muse-xml-encoding-map`
: An alist mapping Emacs coding systems to appropriate XML charsets. Use the base name of the coding system (i.e. without the -unix).

`muse-xml-markup-specials`
: A table of characters which must be represented specially in all XML-like markup formats.

`muse-xml-markup-specials-url-extra`
: A table of characters which must be represented specially in all XML-like markup formats.

  These are extra characters that are escaped within URLs.

`muse-xml-extension`
: Default file extension used for publishing XML files.

`muse-xml-header`
: Header used for publishing XML files.

  This may be text or a filename.

`muse-xml-footer`
: Footer used for publishing XML files.

  This may be text or a filename.

`muse-xml-markup-regexps`
: List of markup rules for publishing a Muse page to XML.

  For more on the structure of this list, See [muse-publish-markup-regexps], page 47.

`muse-xml-markup-functions`
: An alist of style types to custom functions for that kind of text.

  For more on the structure of this list, See [muse-publish-markup-functions], page 47.

`muse-xml-markup-strings`
: Strings used for marking up text.

  These cover the most basic kinds of markup, the handling of which differs little between the various styles.

`muse-xml-encoding-default`
: The default Emacs buffer encoding to use in published files.

  This will be used if no special characters are found.

`muse-xml-charset-default`
: The default XML charset to use if no translation is found in `muse-xml-encoding-map`.

10 Making your own publishing styles

10.1 Specifying functions to mark up text

`muse-publish-markup-functions`

An alist of style types to custom functions for that kind of text.

This is used by publishing styles to attempt to minimize the amount of custom regexps that each has to define. '`muse-publish`' provides rules for the most common types of markup.

Each member of the list is of the following form.

```
(SYMBOL FUNCTION)
```

- SYMBOL Describes the type of text to associate with this rule. `muse-publish-markup-regexps` maps regexps to these symbols.
- FUNCTION Function to use to mark up this kind of rule if no suitable function is found through the '`:functions`' tag of the current style.

10.2 Markup rules for publishing

`muse-publish-markup-regexps`

List of markup rules for publishing a page with Muse.

The rules given in this variable are invoked first, followed by whatever rules are specified by the current style.

Each member of the list is either a function, or a list of the following form.

```
(REGEXP/SYMBOL TEXT-BEGIN-GROUP REPLACEMENT-TEXT/FUNCTION/SYMBOL)
```

- REGEXP A regular expression, or symbol whose value is a regular expression, which is searched for using 're-search-forward'.
- TEXT-BEGIN-GROUP The matching group within that regexp which denotes the beginning of the actual text to be marked up.
- REPLACEMENT-TEXT A string that will be passed to 'replace-match'.

  If it is not a string, but a function, it will be called to determine what the replacement text should be (it must return a string). If it is a symbol, the value of that symbol should be a string.

The replacements are done in order, one rule at a time. Writing the regular expressions can be a tricky business. Note that case is never ignored. 'case-fold-search' is always bound to nil while processing the markup rules.

Publishing order

This is the order that the publishing rules are consulted, by default. This may be changed by customizing `muse-publish-markup-regexps`.

`trailing and leading whitespace`
: Remove trailing and leading whitespace from a file.

directive
: '#directive'

  This is only recognized at the beginning of a file.

comment
: '; a commented line'

tag
: '<tag>'

comment
: '; comment'

explicit links
: Prevent emphasis characters in explicit links from being marked up.

  Don't actually publish them here, just add a special no-emphasis text property.

word
: Whitespace-delimited word, possibly with emphasis characters

  This function is responsible for marking up emphasis and escaping some specials.

heading
: '** Heading'

  Outline-mode style headings.

enddots
: '....'

  These are ellipses with a dot at end.

dots
: '...'

  Ellipses.

rule
: '----'

  Horizontal rule or section separator.

no-break-space
: '~~'

  Prevent lines from being split before or after these characters.

line-break
: '
'

  Break a line at point.

fn-sep
: 'Footnotes:'

  Beginning of footnotes section.

footnote
: '[1]'

  Footnote definition or reference. If at beginning of line, it is a definition.

list
:
  - ' 1. '
  - ' - '
  - 'term :: '

  Numbered list, item list, or term definition list.

table-el
: 'table.el' style tables

`table` ‘`table | cells`’
: Muse tables or orgtbl-mode style tables.

`quote` spaces before beginning of text
: Blockquotes.

`emdash` ‘`--`’
: 2-wide dash

`verse` ‘`> verse text`’

`anchor` ‘`#anchor`’

`link` ‘`[[explicit][links]]`’

`url` ‘`http://example.com/`’

`email` ‘`bare-email@example.com`’

10.3 Strings specific to a publishing style

Markup strings are strings used for marking up text for a particular style.

These cover the most basic kinds of markup, the handling of which differs little between the various styles.

Available markup strings

`image-with-desc`
: An image and a description.

  Argument 1: image without extension. Argument 2: image extension. Argument 3: description.

`image` An inlined image.
: Argument 1: image without extension. Argument 2: image extension.

`image-link`
: An image with a link around it.

  Argument 1: link. Argument 2: image without extension. Argument 3: image extension.

`anchor-ref`
: A reference to an anchor on the current page.

  Argument 1: anchor name. Argument 2: description if one exists, or the original link otherwise.

`url` A URL without a description.
: Argument 1: URL.

`link` A link to a Muse page with a description.
: Argument 1: link. Argument 2: description if one exists, or the original link otherwise.

`link-and-anchor`
: A link to a Muse page with an anchor, and a description.

  Argument 1: link. Argument 2: anchor name. Argument 3: description if one exists, or the original link otherwise. Argument 4: link without an extension.

`email-addr`
: A link to an email address.

  Argument 1: email address. Argument 2: email address.

`anchor`
: An anchor.

  Argument 1: name of anchor.

`emdash`
: A 2-length dash.

  Argument 1: Initial whitespace. Argument 2: Terminating whitespace.

`comment-begin`
: Beginning of a comment.

`comment-end`
: End of a comment.

`rule`
: A horizontal line or space.

`no-break-space`
: A space that separates two words which are not to be separated.

`footnote`
: Beginning of footnote.

`footnote-end`
: End of footnote.

`footnotemark`
: Mark a reference for the current footnote.

  Argument 1: number of this footnote.

`footnotemark-end`
: End of a reference for the current footnote.

`footnotetext`
: Indicate the text of the current footnote.

  Argument 1: number of this footnote.

`footnotetext-end`
: End of a footnote text line.

`fn-sep`
: Text used to replace "Footnotes:" line.

`dots`
: 3 dots.

`enddots`
: 4 dots.

`part`
: Beginning of a part indicator line. This is used by book publishing.

`part-end`
: End of a part indicator line. This is used by book publishing.

`chapter`
: Beginning of a chapter indicator line. This is used by book publishing.

`chapter-end`
: End of a chapter indicator line. This is used by book publishing.

`section`
: Beginning of level 1 section indicator line.

  Argument 1: level of section; always 1.

`section-end`
: End of level 1 section indicator line.

  Argument 1: level of section; always 1.

`subsection`
: Beginning of level 2 section indicator line.

  Argument 1: level of section; always 2.

`subsection-end`
: End of level 2 section indicator line.

  Argument 1: level of section; always 2.

`subsubsection`
: Beginning of level 3 section indicator line.

  Argument 1: level of section; always 3.

`subsubsection-end`
: End of level 3 section indicator line.

  Argument 1: level of section; always 3.

`section-other`
: Beginning of section indicator line, where level is greater than 3.

  Argument 1: level of section.

`section-other-end`
: End of section indicator line, where level is greater than 3.

  Argument 1: level of section.

`begin-underline`
: Beginning of underlined text.

`end-underline`
: End of underlined text.

`begin-literal`
: Beginning of verbatim text. This includes `<code>` tags and =teletype text=.

`end-literal`
: End of verbatim text. This includes `<code>` tags and =teletype text=.

`begin-emph`
: Beginning of the first level of emphasized text.

`end-emph`
: End of the first level of emphasized text.

`begin-more-emph`
: Beginning of the second level of emphasized text.

`end-more-emph`
: End of the second level of emphasized text.

`begin-most-emph`
: Beginning of the third (and final) level of emphasized text.

`end-most-emph`
: End of the third (and final) level of emphasized text.

`begin-verse`
: Beginning of verse text.

`verse-space`
: String used to each space that is further indented than the beginning of the verse.

`begin-verse-line`
: Beginning of a line of verse.

`empty-verse-line`
: End of a line of verse.

`begin-last-stanza-line`
: Beginning of the last line of a verse stanza.

`end-last-stanza-line`
: End of the last line of a verse stanza.

`end-verse`
: End of verse text.

`begin-example`
: Beginning of an example region. To make use of this, an '`<example>`' tag is needed.

`end-example`
: End of an example region. To make use of this, an '`</example>`' tag is needed.

`begin-center`
: Begin a centered line.

`end-center`
: End a centered line.

`begin-quote`
: Begin a quoted region.

`end-quote`
: End a quoted region.

`begin-quote-item`
: Begin a quote paragraph.

`end-quote-item`
: End a quote paragraph.

`begin-uli`
: Begin an unordered list.

`end-uli` End an unordered list.

`begin-uli-item`
Begin an unordered list item.

`end-uli-item`
End an unordered list item.

`begin-oli`
Begin an ordered list.

`end-oli` End an ordered list.

`begin-oli-item`
Begin an ordered list item.

`end-oli-item`
End an ordered list item.

`begin-dl` Begin a definition list.

`end-dl` End a definition list.

`begin-dl-item`
Begin a definition list item.

`end-dl-item`
End a definition list item.

`begin-ddt`
Begin a definition list term.

`end-ddt` End a definition list term.

`begin-dde`
Begin a definition list entry.

`end-dde` End a definition list entry.

`begin-table`
Begin a table.

`end-table`
End a table.

`begin-table-group`
Begin a table grouping.

`end-table-group`
End a table grouping.

`begin-table-row`
Begin a table row.

`end-table-row`
End a table row.

`begin-table-entry`
Begin a table entry.

`end-table-entry`
End a table entry.

10.4 Tag specifications for special markup

`muse-publish-markup-tags`

A list of tag specifications, for specially marking up text.

XML-style tags are the best way to add custom markup to Muse. This is easily accomplished by customizing this list of markup tags.

For each entry, the name of the tag is given, whether it expects a closing tag and/or an optional set of attributes, whether it is nestable, and a function that performs whatever action is desired within the delimited region.

The tags themselves are deleted during publishing, before the function is called. The function is called with three arguments, the beginning and end of the region surrounded by the tags. If properties are allowed, they are passed as a third argument in the form of an alist. The 'end' argument to the function is always a marker.

Point is always at the beginning of the region within the tags, when the function is called. Wherever point is when the function finishes is where tag markup will resume.

These tag rules are processed once at the beginning of markup, and once at the end, to catch any tags which may have been inserted in-between.

10.5 Parameters used for defining styles

Style elements are tags that define a style. Use either `muse-define-style` or `muse-derive-style` (see Section 10.6 [Deriving Styles], page 55) to create a new style.

`muse-define-style` *name* **&rest** *elements* [Function]

Usable elements

'`:suffix`' File extension to use for publishing files with this style.

'`:link-suffix`'
File extension to use for publishing links to Muse files with this style.

'`:osuffix`'
File extension to use for publishing second-stage files with this style.

For example, PDF publishing generates a LaTeX file first, then a PDF from that LaTeX file.

'`:regexps`'
List of markup rules for publishing a page with Muse. See [muse-publish-markup-regexps], page 47.

'`:functions`'
An alist of style types to custom functions for that kind of text. See [muse-publish-markup-functions], page 47.

'`:strings`'
Strings used for marking up text with this style.

These cover the most basic kinds of markup, the handling of which differs little between the various styles.

‘:tags’
: A list of tag specifications, used for handling extra tags. See [muse-publish-markup-tags], page 54.

‘:specials’
: A table of characters which must be represented specially.

‘:before’
: A function that is to be executed on the newly-created publishing buffer (or the current region) before any publishing occurs.

  This is used to set extra parameters that direct the publishing process.

‘:before-end’
: A function that is to be executed on the publishing buffer (or the current region) immediately after applying all of the markup regexps.

  This is used to fix the order of table elements (header, footer, body) in XML-ish styles.

‘:after’
: A function that is to be executed on the publishing buffer after :before-end, and immediately after inserting the header and footer.

  This is used for generating the table of contents as well as setting the file coding system.

‘:final’
: A function that is to be executed after saving the published file, but while still in its buffer.

  This is used for generating second-stage documents like PDF files from just-published LaTeX files.

  The function must accept three arguments: the name of the muse source file, the name of the just-published file, and the name of the second-stage target file. The name of the second-stage target file is the same as that of the just-published file if no second-stage publishing is required.

‘:header’
: Header used for publishing files of this style.

  This may be a variable, text, or a filename. It is inserted at the beginning of a file, after evaluating the publishing markup.

‘:footer’
: Footer used for publishing files of this style.

  This may be a variable, text, or a filename. It is inserted at the end of a file, after evaluating the publishing markup.

‘:style-sheet’
: Style sheet used for publishing files of this style.

  This may be a variable or text. It is used in the header of HTML and XHTML based publishing styles.

‘:browser’
: The function used to browse the published result of files of this style.

10.6 Deriving a new style from an existing one

To create a new style from an existing one, use `muse-derive-style` as follows. This is a good way to fix something you don’t like about a particular publishing style, or to personalize it.

`muse-derive-style` *new-name base-name* **&rest** *elements* [Function]

The derived name is a string defining the new style, such as "my-html". The base name must identify an existing style, such as "html" – if you have loaded '`muse-html`'. The style parameters are the same as those used to create a style, except that they override whatever definitions exist in the base style. However, some definitions only partially override. The following parameters support partial overriding.

See Section 10.5 [Style Elements], page 54, for a complete list of all parameters.

'`:functions`'
: If a markup function is not found in the derived style's function list, the base style's function list will be queried.

'`:regexps`'
: All regexps in the current style and the base style(s) will be used.

'`:strings`'
: If a markup string is not found in the derived style's string list, the base style's string list will be queried.

11 Miscellaneous add-ons, like a minor mode

11.1 Edit lists easily in other major modes

`muse-list-edit-minor-mode` is meant to be used with other major modes, such as Message (for composing email) and debian-changelog-mode (for editing debian/changelog files).

It implements practically perfect support for editing and filling lists. It can even handle nested lists. In addition to Muse-specific list items (`"-"`, numbers, definition lists, footnotes), it can also handle items that begin with `"*"` or `"+"`. Filling list items behaves in the same way that it does in Muse, regardless of whether filladapt is also enabled, which is the primary reason to use this tool.

Installation

To use it, add "(require 'muse-mode)" to your Emacs customization file and add the function `turn-on-muse-list-edit-minor-mode` to any mode hooks where you wish to enable this minor mode.

Keybindings

`muse-list-edit-minor-mode` uses the following keybindings.

M-RET ('muse-l-e-m-m-insert-list-item')
: Insert a new list item at point, using the indentation level of the current list item.

C-< ('muse-l-e-m-m-decrease-list-item-indent')
: Decrease indentation of the current list item.

C-> ('muse-l-e-m-m-increase-list-item-indent')
: Increase indentation of the current list item.

Functions

`muse-list-edit-minor-mode` [Function]

This is a global minor mode for editing files with lists. It is meant to be used with other major modes, and not with Muse mode.

Interactively, with no prefix argument, toggle the mode. With universal prefix *arg* turn mode on. With zero or negative *arg* turn mode off.

This minor mode provides the Muse keybindings for editing lists, and support for filling lists properly.

It recognizes not only Muse-style lists, which use the `"-"` character or numbers, but also lists that use asterisks or plus signs. This should make the minor mode generally useful.

Definition lists and footnotes are also recognized.

Note that list items may omit leading spaces, for compatibility with modes that set `left-margin`, such as `debian-changelog-mode`.

`turn-on-muse-list-edit-minor-mode` [Function]

Unconditionally turn on Muse list edit minor mode.

`turn-off-muse-list-edit-minor-mode` [Function]

Unconditionally turn off Muse list edit minor mode.

12 Getting Help and Reporting Bugs

After you have read this guide, if you still have questions about Muse, or if you have bugs to report, there are several places you can go.

- `http://www.emacswiki.org/cgi-bin/wiki/EmacsMuse` is the emacswiki.org page, and anyone may add tips, hints, or bug descriptions to it.
- `http://mwolson.org/projects/EmacsMuse.html` is the web page that Michael Olson (the current maintainer) made for Muse.
- Muse has several different mailing lists.

‘`muse-el-announce`’
: Low-traffic list for Muse-related announcements.

  You can join this mailing list (`muse-el-announce@gna.org`) using the subscription form at `http://mail.gna.org/listinfo/muse-el-announce/`. This mailing list is also available via Gmane (`http://gmane.org/`). The group is called ‘`gmane.emacs.muse.announce`’.

‘`muse-el-discuss`’
: Discussion, bugfixes, suggestions, tips, and the like for Muse. This mailing list also includes the content of muse-el-announce.

  You can join this mailing list (`muse-el-discuss@gna.org`) using the subscription form at `http://mail.gna.org/listinfo/muse-el-discuss/`. This mailing list is also available via Gmane with the identifier ‘`gmane.emacs.muse.general`’.

‘`muse-el-logs`’
: Log messages for commits made to Muse.

  You can join this mailing list (`muse-el-logs@gna.org`) using the subscription form at `http://mail.gna.org/listinfo/muse-el-logs/`. This mailing list is also available via Gmane with the identifier ‘`gmane.emacs.muse.scm`’.

‘`muse-el-commits`’
: Generated bug reports for Emacs Muse. If you use our bug-tracker at `https://gna.org/bugs/?group=muse-el`, the bug reports will be sent to this list automatically.

  You can join this mailing list (`muse-el-commits@gna.org`) using the subscription form at `http://mail.gna.org/listinfo/muse-el-commits/`. This mailing list is also available via Gmane with the identifier ‘`gmane.emacs.muse.cvs`’.

‘`muse-el-internationalization`’
: Discussion of translation of the Muse website and documentation into many languages.

  You can join this mailing list (`muse-el-internationalization@gna.org`) using the subscription form at `http://mail.gna.org/listinfo/internationalization/`. This mailing list is also available via Gmane with the identifier ‘`gmane.emacs.muse.internationalization`’.

- You can visit the IRC Freenode channel '`#emacs`'. Many of the contributors are frequently around and willing to answer your questions. The '`#muse`' channel is also available for Muse-specific help, and its current maintainer hangs out there.
- The maintainer of Emacs Muse, Michael Olson, may be contacted at `mwolson@gnu.org`. He can be rather slow at answering email, so it is often better to use the muse-el-discuss mailing list.

13 History of This Document

- 2004 John Wiegley started Muse upon realizing that EmacsWiki had some serious limitations. Around February 2004, he started making "emacs-wiki version 3.00 APLHA", which eventually became known as Muse.

  Most of those who frequent the emacs-wiki mailing list continued to use emacs-wiki, mainly because Planner hasn't been ported over to it.

  As of 2004-12-01, Michael Olson became the maintainer of Muse, as per John Wiegley's request.

- 2005 Michael Olson overhauled this document and added many new sections in preparation for the first release of Muse (3.01).

14 Contributors to This Documentation

The first draft of this document was taken from the emacs-wiki texinfo manual. Michael Olson adapted it for Muse and added most of its content.

John Sullivan did a majority of the work on the emacs-wiki texinfo manual.

While Sacha Chua maintained emacs-wiki, she worked quite a bit on the emacs-wiki texinfo manual.

Appendix A GNU Free Documentation License

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.

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

10. FUTURE REVISIONS OF THIS LICENSE

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

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

ADDENDUM: How to use this License for your documents

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

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
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.

Index

P

Q

R

S

T

U

V

W