The Emacs Muse

John Wiegley and Michael Olson

June 17, 2007

Emacs Muse is an authoring and publishing environment for Emacs. It simplifies the process of writings 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.

1 About this document

This document provides an example of Muse markup and also functions as a quickstart for Muse.

To see what it looks like when published, type `make examples`. You will then get an Info document, an HTML document, and a PDF document (provided you have an implementation of LaTeX installed with the necessary fonts).

2 Getting Started

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. For example:

```
(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
```

Once loaded, the command `M-x muse-publish-this-file` will publish an input document to any available style. If you enable `muse-mode` within a buffer, by typing `M-x muse-mode`, this command will be bound to `C-c C-t`.

3 Creating a Muse project

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 creations of "projects". Here is a sample project, to 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]]`.

4 Markup rules

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.

4.1 Paragraphs

Separate paragraphs in Muse must be separate by a blank line.

For example, the input text used for this section is:

```
Separate paragraphs in Muse must be separate by a blank line.

For example, the input text used for this section is:
```

4.2 Centered paragraphs and quotations

A line that begins with six or more columns of whitespace (either tabs or spaces) indicates a centered paragraph.

This is centered

> But if a line begins with whitespace, though less than six columns, it indicates a quoted paragraph.

4.3 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 to three asterices, followed by a space and the heading title. Then begin another paragraph to enter the text for that section.

```
* First level

** Second level

*** Third level
```

4.4 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!

The separator above was produced by typing:

```
----
```

4.5 Emphasizing text

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

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

The above list renders as:

emphasis
strong emphasis
very strong emphasis
underlined
`verbatim and monospace`

4.6 Adding footnotes

A footnote reference is simply a number in square brackets[1].[1] 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.

[1] This is a footnote.

```
Footnotes:
[1]  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.
```

4.7 Verse

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

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

The above is rendered as:

> 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>
```

4.8 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 format, use the `style` attribute for the `<literal>` tag. Some examples follow.

```
You are reading the
<literal style="html">HTML</literal>
<literal style="pdf">PDF</literal>
<literal style="info">Info</literal>
version of this document.
```

Produces:
You are reading the PDF version of this document.

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

Produces:
LaTeX was used in the publishing of this document.

4.9 Lists

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.

The supported kinds of lists are:

```
    - bullet item one
    - bullet item two

    1. Enumerated item one
    2. Enumerated item two

  Term1 :: A definition one

  Term2 :: A definition two
```

These are rendered as a bullet list:

- bullet item one
- bullet item two

An enumerated list:

1. Enum item one
2. Enum item two

And a definition list:

Term1

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

Term2

: This is a second definition

Lists may be nested inside of one another. The level of nesting is determined by the amount of leading whitespace.

```
  - Level 1, bullet item one
    1. Level 2, enum item one
    2. Level 2, enum item two
  - Level 1, bullet item two
```

This is rendered as:

- Level 1, bullet item one

    1. Level 2, enum item one
    2. Level 2, enum item two

- Level 1, bullet item two

4.10 Tables

Simple tables are supported. The syntax is:

```
Double bars  || Separate header fields

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

Triple bars ||| Separate footer fields
```

The above is rendered as:

Double bars	Separate header fields
Single bars	Separate body fields
Here are more	body fields
Triple bars	Separate footer fields

It is also possible to make tables that look like:

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

This publishes as:

Single bars	Separate body fields
Here are more	body fields

If you are familiar with Org-mode style tables, simple variants (no column groups or autogenerated formulas) will publish fine. Also, table.el style tables will be published, provided that the output format is something that table.el supports.

4.11 Anchors and tagged links

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 (see below).

Click **??** to go back to the previous paragraph.

4.12 URLs and E-mail addresses

A URL or e-mail address encountered in the input text is published as a hyperlink if the output style supports it. For example, the latest Muse source can be downloaded at `http://download.gna.org/muse-el` and mail may be sent to `mwolson@gnu.org`.

4.13 Images

If a URL has an extension of a known image format, it will be inlined if possible. To create a link to an image, rather than inlining it, put the text "URL:" immediately in front of the link.

This is an inlined image example:

```
Made with [[muse-made-with.png]] Emacs Muse.
```

This publishes as:
Made with Emacs Muse.

Here is an example of a captioned image:

```
[[emacs-muse.png][Muse, the publishing choice of Great Thinkers]]
```

This publishes as:

Figure 1: Emacs Muse, the publishing choice of (a subset of) Great Thinkers

The following will be published as a link only.

```
The Muse logo: [[URL:http://mwolson.org/static/logos/emacs-muse.png]].
```

The Muse logo: `http://mwolson.org/static/logos/emacs-muse.png`.

4.14 Links

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:

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

Thus, the text:

```
Muse can be downloaded [[http://download.gna.org/muse-el/][here]], or at
[[http://download.gna.org/muse-el/]].
```

Publishes as:
Muse can be downloaded here[2], or at `http://download.gna.org/muse-el/`.

4.15 Source code

If you have htmlize.el version 1.34 or later installed, you can publish colorized HTML for source code in any major mode that Emacs supports by using the `<src>` tag. If not publishing to HTML, the text between the tags will be treated like an `<example>` tag.

Here is some example C code. Muse takes the `lang` element and appends `"-mode"` to it in order to determine which major mode to use when colorizing the code.

```
<src lang="c">
#include <stdlib.h>

char *unused;

int main (int argc, char *argv[])
{
    puts("Hello, world!\n");
}
</src>
```

Here is the colorized output. This may look different if you have customized some faces.

```
#include <stdlib.h>

char *unused;
```

[2] http://download.gna.org/muse-el/

```
int main (int argc, char *argv[3])
{
    puts("Hello, world!\n");
}
```

4.16 Embedded Lisp, Perl, Ruby, Python, or Shell

Arbitrary kinds of markup can be achieved using the `<lisp>` tag, which is the only Muse tag supported in a style's header and footer text. With the `<lisp>` tag, you may generated 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>
```

The above renders as: This form gets inserted.

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 tag.

For example:

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

The output is:

```
Insert me
```

This `markup` attribute can also be passed to the `<perl>`, `<ruby>`, `<python>`, and `<command>` tags, which interpret Perl, Ruby, Python, and Shell code, respectively.

5 Publishing styles

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.

5.1 Deriving from an existing style

To create a new style from an existing one, use `muse-derive-style`:

```
(muse-derive-style DERIVED-NAME BASE-NAME STYLE-PARAMETERS)
```

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.

Most often, this will come in handy for using a custom header, footer, and/or stylesheet for a project. Here is one such example.

```
(setq my-different-style-sheet
      (concat "<link rel=\"stylesheet\" type=\"text/css\""
              " charset=\"utf-8\" media=\"all\""
              " href=\"/different.css\" />"))

(muse-derive-style "my-xhtml" "xhtml"
                   :header "~/.emacs.d/muse/different-header.html"
                   :footer "~/.emacs.d/muse/different-footer.html"
                   :style-sheet my-different-style-sheet)
```

Many parameters support being partially overridden. As an example: with `:functions`, if a markup function is not found in the derived style's function list, the base style's function list will be queried.

If a parameter takes the name of a function, then returning non-nil from that function causes no further processing to be done. If the function returns nil, any other functions in the base style will be called.

5.2 Creating a new style

To create a new style, use `muse-define-style`:

```
(muse-define-style NAME STYLE-PARAMETERS)
```

If you want to create a new style, it is best to examine the source code for other styles first, to get an idea of what needs to be done. Each output format should have its own file, containing all styles based on it. For example. the `latex`, `latex-slides`, and `pdf` styles are all contained in `muse-latex.el`.

If you are willing to sign copyright papers for the Free Software Foundation (we will help you with this step), the Muse authors may be interested in including your work in future versions of Muse.

6 License

Another footnote, this one unreferenced.