vfmd

Markdown with a spec

The vfmd syntax guide

vfmd is a variant of Markdown with an unambiguous specification of its syntax.

This document describes the vfmd syntax in simple terms, and is intended to be of help for people writing in vfmd.

The vfmd syntax is pretty much the same as the original Markdown syntax, with some differences and additional details.

vfmd enables you to represent rich-text markup in plain-text files, with an obsessive emphasis on readability. The intention is to keep the plain text representation publishable as-is, without looking like it's been marked up with tags or formatting instructions.

vfmd supports block-level markup like lists, blockquotes and code-blocks, and also supports span-level markup like emphasis, links and inline-code.

Table of contents

Basics

Encoding

The input text should be in UTF-8 encoding. If you are writing your vfmd document in English, it is most likely in UTF-8. If you are writing in another language, please make sure that the document is in UTF-8 encoding (for example, if you are using a text editor to write the document, ensure that it saves the document in UTF-8 encoding).

Tabs

vfmd assumes 4-column tab stops when dealing with tab characters in the input. If you are using a text editor to write the text, please configure your editor to either use 4-column tab stops, or to expand tabs to spaces as you type.

Block-level elements

Paragraphs

A paragraph is one or more consecutive lines of text, separated by one or more blank lines. (A blank line is any line that looks like a blank line

— a line containing nothing but spaces or tabs is considered blank.) Normal paragraphs should not be indented with spaces or tabs.

To introduce a hard line-break in the middle of a paragraph, you end a line with two or more spaces, then type return.

Headers

vfmd supports both setext-style and atx-style headers.

setext-style headers

To create a setext-style header, follow the header text with a line that "underlines" the text with equal signs (`=`) or dashes (`-`).

For a first-level header, the "underline" line should be sequence of one or more `=` characters. For a second-level header, it should be a sequence of one or more `-` characters. There can be no other characters in the "underline" line (trailing spaces are okay).

For example:

```
This is a first-level header
============================

This is a second-level header
-----------------------------

This is a first-level header
===

This is a second-level header
---
```

Please note that in case the header is in the immediate next line of a paragraph, it will be considered as a part of the paragraph, and shall not be identified as a header.

```
This is a paragraph. If you don't inculde a blank line
between this paragraph of text and the title of the next
section, the title will not be recognized as a title.
This will not be recognized as a section title
----------------------------------------------

If you do include a blank line between the paragraph and
the title, it all works out correctly.

This is a section title
-----------------------
```

atx-style headers

atx-style headers start with one or more `#` characters. The first-level header starts with a single `#`, second-level headers with `##`, third-level headers with `###`, and so on, till 6 `#` characters for a sixth-level header.

For example:

```
# Title (first-level header)

## Section (second-level header)

### Subtitle (third-level header)

###### Sixth-level header

######### Still a sixth-level header
```

You may optionally "close" the header with any number of additional `#` characters. However, the header level is determined only by counting the "opening" `#` characters.

For example:

```
## Second-level header ##

### Third-level header ######

###### Sixth-level header #
```

Please note that in case the header is in the immediate next line of a paragraph, it will be considered as a part of the paragraph, and shall not be identified as a header.

```
This is a paragraph. If you don't inculde a blank line
between this paragraph of text and the title of the next
section, the title will not be recognized as a title.
# This will not be recognized as a section title

If you do include a blank line between the paragraph and
the title, it all works out correctly.

# This is a section title
```

Code blocks

Code blocks can be used to quote text verbatim. For example, it can be used to quote source code. Every line of the code block should be indented by 4 space characters.

No Markdown syntax is processed within a code block. In addition, for HTML output, ampersands (`&`) and angle brackets (`<` and `>`) within a code block are automatically converted into HTML entities, so that the text appears as specified when the output HTML in viewed in a browser.

For example:

```
Consider the following snippet that mixes Markdown with HTML5:

    ## Introduction

    This is <del>normal text</del> an intro.
```

becomes, in HTML output:

```
<p>Consider the following snippet that mixes Markdown with HTML5:</

<pre><code>## Introduction

This is &lt;del&gt;normal text&lt;/del&gt; an intro.
</pre></code>
```

If you'd like to have two code blocks one after the other, please use two blank lines to separate them.

Code blocks should be separated from other block-level elements with a blank line.

Blockquotes

Blockquoting is done by starting a line with `>` characters, like it's done in email. It looks best if you hard wrap the text and start every line with a `>`, like this:

```
> 'Very true,' said the Duchess: 'flamingoes and mustard both
> bite.  And the moral of that is - "Birds of a feather flock
> together."'
>
> 'Only mustard isn't a bird,' Alice remarked.
>
> 'Right, as usual,' said the Duchess: 'what a clear way you
> have of putting things!'
```

But you are allowed to skip the leading `>` for subsequent lines of a hard-wrapped paragraph, as long as the first line of the paragraph starts with a `>`. If there's only one blank line between multiple blockquoted paragraphs, they form a single blockquote, like this:

```
> 'Very true,' said the Duchess: 'flamingoes and mustard both
bite.  And the moral of that is - "Birds of a feather flock
together."'

> 'Only mustard isn't a bird,' Alice remarked.

> 'Right, as usual,' said the Duchess: 'what a clear way you
have of putting things!'
```

If you'd like to have two blockquotes one after the other, please use two blank lines to separate them.

Blockquotes can contain other elements, like headers, code blocks and nested blockquotes.

For example:

```
> ## Section header
>
> This is a paragraph inside a blockquote.
>
> > This is a nested blockquote.
> > The second line of the nested blockquote.
> > > This is the third level of nesting.
>
> Some code that gives the ultimate answer of
> _Life, the Universe and Everything_:
>
>     int main() {
>         return 42;
>     }
```

Blockquotes should be separated from other block-level elements with a blank line. However, a preceding blank line is not necessary for nested blockquotes.

For example:

```
This text should be followed by a blank line
before the blockquote can begin.

> But within the blockquote, nested blockquotes
> can start without being separated by a blank
> line.
> > This is a nested blockquote
> > that starts without a preceding blank line
```

Lists

A numbered list is created by numbering each item in the list, like this:

```
 1. Donuts
 2. Chocolate
 3. Cupcakes
```

The numbers used in the list need not necessarily be ordered - the output will have them in the right order anyway. However, the number used for the first item will be used as the starting number for the list.

A bulleted list is created by using asterisks(`*`) or pluses (`+`) or hyphens (`-`) as list marker characters, like this:

```
  * Donuts
  * Chocolate
  * Cupcakes
```

The list marker character can be placed at the left margin, or may be indented by up to three spaces, and must be followed by one or more spaces.

When you change the list marker character, or change the number of spaces before/after the list marker, each set of uniform list markers form a separate list. For example, the following forms four top-level lists placed one after the other:

```
 *  Donuts
 *  Chocolate
 *  Cupcakes
 +    Banana
 +    Apple
 +    Kiwi
   + The Fellowship of the Ring
   + The Two Towers
   + The Return of the King
 -   Tomato
 -   Cucumber
```

If a list item is separated by blank lines on both sides, the HTML output shall wrap that item with `<p>` tags.

For example, this input:

```
* Bird
* Plane
* Superman
* UFO
* Paper plane
* Dragonfly
* Helicopter
```

will turn into:

```
<ul>
<li>Bird</li>
<li>Plane</li>
<li>Superman</li>
<li>UFO</li>
<li>Paper plane</li>
<li>Dragonfly</li>
<li>Helicopter</li>
</ul>
```

But this:

```
* Bird
* Plane

* Superman

* UFO

* Paper plane
* Dragonfly
* Helicopter
```

will turn into:

```
<ul>
<li>Bird</li>
<li>Plane</li>
<li><p>Superman</p></li>
<li><p>UFO</p></li>
<li>Paper plane</li>
<li>Dragonfly</li>
<li>Helicopter</li>
</ul>
```

If every list item is separated by blank lines, then each of them will get wrapped in `<p>` tags in the HTML output.

Each list item can contain multiple paragraphs. To make lists look nice, you can indent all lines within each list item to vertically align with the starting paragraph of the list item, like this:

```
*   Lorem ipsum dolor sit amet, consectetur adipiscing elit.

    Ut pulvinar, odio a luctus molestie, sapien libero
    hendrerit risus, a mollis quam ipsum non purus.

    Integer dignissim faucibus metus a aliquet. Nulla vitae
    neque mattis, cursus ipsum a, bibendum lorem. Phasellus
    dignissim neque nec libero sollicitudin, nec tempor orci
    vehicula.

*   Vivamus non mauris massa. Aliquam hendrerit feugiat
    vulputate. Ut quis justo dapibus, tempor sem ut,
    ullamcorper sem.
```

But if you want to be lazy, you don’t have to, as long as the first lines of the paragraphs are indented to align vertically:

```
*   Lorem ipsum dolor sit amet, consectetur adipiscing elit.

    Ut pulvinar, odio a luctus molestie, sapien libero
hendrerit risus, a mollis quam ipsum non purus.

    Integer dignissim faucibus metus a aliquet. Nulla vitae
neque mattis, cursus ipsum a, bibendum lorem. Phasellus
dignissim neque nec libero sollicitudin, nec tempor orci
vehicula.

*   Vivamus non mauris massa. Aliquam hendrerit feugiat
vulputate. Ut quis justo dapibus, tempor sem ut, ullamcorper
sem.
```

A list item can contain other block-level elements, like blockquotes, headers, code blocks and nested lists. Any nested element should be indented to align vertically with the starting paragraph of the list item.

```
-   List item with a blockquote. The blockquote should be indented
    to align with the starting position of this paragraph.

    > This is a blockquote
    > inside a list item

-   List item with a code-block. The code-block should be indented
    by 4 spaces from the starting position of this paragraph.

        int main() {
            return 42;
        }

-   List item with two nested lists. The lists can be indented
    by 0 to 3 spaces from the starting position of this paragraph.

     - First
     - Second
     - Third

     1. One
     2. Two
     3. Three
```

You can force-end a list with two consecutive blank lines. You can use this technique to write two numbered lists one after the other, or to have a code-block immediately after a list.

Lists should be separated from other block-level elements with a blank line. However, a preceding blank line is not necessary for nested lists.

For example:

```
This text should be followed by a blank line
before the list can begin.

 *  But within the list, nested lists can
    start without being separated by a
    blank line.
     * Item 1 of nested list
       1. Second level of nesting
       2. Still second level of nesting
     * Item 2 of nested list
     * Item 3 of nested list
```

Horizontal rule

A horizontal rule tag (`<hr/>`) can be created by placing three or more hyphens (`-`), or three or more underscores (`_`) or three or more asterisks (`*`) on a line by themselves. The hyphens (or the underscores, or the asterisks) forming the horizontal rule can optionally be interspersed with spaces.

Each of the following lines results in a horizontal rule:

```
***

*****

* * * *

-------------

-   -   -   -   -   -   -

-- --- --
```

But the following lines don't result in a horizontal rule:

```
**

--

*-*-*
```

```
_-_-_-_
```

Span-level elements

Links

vfmd supports two styles of links: *referenced links* and *inline links*.

Referenced links

In referenced links, the link text is delimited by [square brackets]. For each such link text, the URL is specified in a link definition at the end of the paragraph or section (or actually, anywhere in the document), like this:

```
Nowadays, you can [Google] for pretty
much anything and get answers super quick.
There are even [math engines] and
[language translators] available
for anyone to use.

[Google]: http://www.google.com/
[math engines]: http://www.wolframalpha.com/
[language translators]: http://www.bing.com/translator
```

The link text in the paragraph will become a link pointing to the URL specified at the end of the paragraph. For the above input, the HTML output will be:

```
<p>Nowadays, you can <a href="http://www.google.com/">Google</a> fo
much anything and get answers super quick.
There are even <a href="http://www.wolframalpha.com/">math engines<
<a href="http://www.bing.com/translator">language translators</a> a
for anyone to use.</p>
```

You can also specify an alternate label for referencing the link, by specifying it immediately after the link text, within [square brackets]. This is especially useful if you want to have the same link text linking to different URLs at different points in the document.

For example:

```
You can [search for it][goog-search] in Google,
or [search for it][bing-search] in Bing.
```

```
[goog-search]: http://www.google.com/search?q=something
[bing-search]: http://www.bing.com/search?q=something
```

becomes, in HTML output:

```
<p>You can <a href="http://www.google.com/search?q=something">searc
or <a href="http://www.bing.com/search?q=something">search for it</
```

You can also specify a link title in the link definition. The title should be placed immediately after the URL, surrounded by quotes or parentheses. The title can also be moved to the next line, but it should then be indented with spaces or tabs (which is especially useful when the URL is too long).

The following four link definitions are equivalent:

```
[foo]: http://example.com/  "Optional Title Here"
[foo]: http://example.com/  'Optional Title Here'
[foo]: http://example.com/  (Optional Title Here)
[foo]: http://example.com/
       "Optional Title Here"
```

You can use backslash escapes within a title to include instances of the enclosing character in the title. For example:

```
[foo]: http://example.com/  "Title \"with quotes\" Here"
```

To summarize, a link definition consists of:

- The link identifier enclosed in square brackets (optionally indented from the left margin by up to three spaces);
- followed by a colon;
- followed by one or more spaces (or tabs);
- followed by the URL for the link;
- optionally followed by a title attribute for the link, enclosed in double quotes or single quotes or parentheses (either in the same line as the link identifier and the URL, or indented and present in full in the next line).

The link identifier is not case-sensitive, so the following is valid:

```
Just ask [Google].

[google]: http://google.com/
```

The URL specified in the link can be an absolute URL or a relative URL. The URL can optionally be enclosed in <angle brackets>.

vfmd also supports the *implicit link name* syntax from the original Markdown for backward-compatibility. For example, both the sentences in the following paragraph translate to the same output:

```
Just ask [Google].
Just ask [Google][].

[Google]: http://google.com/
```

Link definitions are only used for defining the links, and don't appear in the output.

If you want to include literal `[` or `]` characters in the link text or in the link identifier, you should escape each such `[` or `]` character with a backslash, like this:

```
"Off with his head", [she \[the Queen\] said].

[she \[the Queen\] said]: http://example.net/queen_of_hearts_said/
```

Similarly, if you want to include literal `` ` `` (backtick) characters in the link text, link identifier or the link title, you should escape each such backtick character with a backslash. Also note that the link URL should not have any URL-unsafe characters like `<`, `>` or `` ` `` - if required, they should be escaped, for example, to `&lt;`, `&gt;` or `&#96` respectively.

Inline links

In inline links, the link text is delimited by [square brackets], and is immediately followed by a set of (regular parentheses). Inside the parentheses, you put the URL where you want the link to point, along with an optional title for the link, surrounded in quotes. For example:

```
This is [an example](http://example.com/ "Title") inline link.

[This link](http://example.net/) has no title attribute.
```

will produce the HTML output:

```
<p>This is <a href="http://example.com/" title="Title"> an example<

<p><a href="http://example.net/">This link</a> has no title attribu
```

The link URL can also be a relative URL. For example:

```
See my [About](/about/) page for details, or [contact me](#contact)
```

You can optionally enclose the URL in <angle brackets>. However, in case your URL contains the `(` or `)` characters, the angle brackets are mandatory. For example:

```
Time-travel is hard to get right and Pratchett does a great job of
it in [Nightwatch](<http://en.wikipedia.org/wiki/Night_Watch_(Discw
```

If your URL is too long and you'd like to break it up across multiple lines, then again the angle brackets are mandatory. For example:

```
There are even [children's books](<http://en.wikipedia.org/wiki/The
Amazing_Maurice_and_his_Educated_Rodents> "Amazing Maurice book")
set in the Discworld.
```

Like in referenced links, if you want to include any `[` or `]` or `` ` `` characters in the link text, or any quote characters in the title text, you should escape them with backslashes, like this:

```
Go to [step \[1\]](#foo-step-1 "Step 1 of the \"foo\" routine")
```

Also, the link URL should not have any URL-unsafe characters like `<`, `>` or `` ` `` - if required, they should be escaped, for example, to `&lt;`, `&gt;` or `&#96` respectively.

Emphasis

To emphasize text, you use asterisks (`*`) or underscores (`_`). Text wrapped with one `*` or `_` will be given *emphatic stress* (using `<em>` tags in HTML output); text wrapped in double `*`s or double `_`s will be output as *strong text* (using `<strong>` tags in HTML output).

For example, this input:

```
*single asterisks*

_single underscores_

**double asterisks**
```

```
__double underscores__
```

will produce the HTML output:

```
<em>single asterisks</em>

<em>single underscores</em>

<strong>double asterisks</strong>

<strong>double underscores</strong>
```

You can use whichever style you prefer; the lone restriction is that the same character must be used to open and close an emphasis span.

Emphasis cannot be used in the middle of a word; `*` or `_` characters appearing in the middle of a word will be treated as literal asterisks or underscores. Similarly, `*` or `_` characters surrounded by spaces will be treated as literal asterisks or underscores.

For example:

```
There is no ** emphasis ** in_this_sentence.
```

To produce a literal asterisk or underscore at a position where it would otherwise be used as an emphasis delimiter, you can backslash escape it:

```
\*this text is surrounded by literal asterisks\*
```

Code

To indicate a span of code, wrap it with backtick quotes (`` ` ``). Unlike a pre-formatted code block, a code span indicates code within a normal paragraph. For example:

```
Use the `printf()` function.
```

will produce the HTML output:

```
<p>Use the <code>printf()</code> function.</p>
```

To include a literal backtick character within a code span, you can use multiple backticks as opening and closing delimiters. For example:

```
``There is a literal backtick (`) here.``
```

will become, in HTML:

~~~
<p><code>There is a literal backtick (`) here.</code></p>
~~~

More generally, to include multiple backticks in the code span, you should use a different number of backticks as opening and closing delimiters. For example, if you want a code span containing both a single-backtick sequence and a double-backtick sequence, you can use three (or four, if you prefer) backticks as delimiters:

~~~
```One (`) and two (``) backticks```
~~~

To start or end a code span with a literal backtick, include one or more spaces between the code span and the delimiting backticks. For example:

~~~
A single backtick in a code span: `` ` ``

A backtick-delimited string in a code span: `` `foo` ``
~~~

will produce:

~~~
<p>A single backtick in a code span: <code>`</code></p>

<p>A backtick-delimited string in a code span: <code>`foo`</code></
~~~

Within a code span, ampersands and angle brackets are encoded as HTML entities automatically, which makes it easy to include HTML tags in a code span. For example:

~~~
Please don't use any `<blink>` tags
or `&nbsp;` spaces.
~~~

will become, in HTML output:

~~~
<p>Please don't use any <code>&lt;blink&gt;</code> tags
or <code>&amp;nbsp;</code> spaces.</p>
~~~

You can use a backslash to escape a backtick to prevent it from starting a code span. However, within a code span, a backslash is treated as a literal backslash and not as an escaping character. This is so that when you cut-and-paste code, you wouldn't have to edit inside the pasted code to escape any backslashes in it.

Images

The syntax for images is very similar to the syntax for links - the only addition is an exclamation mark. Like links, there are two styles in which you can include images: *referenced* and *inline*.

Referenced images

In the referenced-image syntax, you type an exclamation mark followed by an identifying label for the image enclosed in [square brackets]. For each such label, the image URL is specified elsewhere in the document (very much like how link URLs are specified in link definitions). The image definition can optionally include a title, enclosed in quotes.

For example:

```
![sleeping kitten]

Above: A kitten taking a nap

[sleeping kitten]: path/to/sleeping_kitten.jpg "Optional title"
```

The identifying label for the image will be used as the `alt` text in HTML output. For the above example, the HTML output would be:

```
<p><img src="path/to/sleeping_kitten.jpg" alt="sleeping kitten" tit

<p>Above: A kitten taking a nap</p>
```

If you want the same `alt` text for different images in the document, you can specify a different label for referencing the image:

```
![sleeping kitten][kitten-1]
![sleeping kitten][kitten-2]

Above: Kittens taking a nap

[kitten-1]: path/to/sleeping_kitten_1.jpg
[kitten-2]: path/to/sleeping_kitten_2.jpg
```

You can also make the image point to a URL by placing it as the link content of a link, like this:

```
[![sleeping kitten]]

(Click to see a larger version of the image)

[sleeping kitten]: /thumbnails/sleeping_kitten.jpg
[![sleeping kitten]]: /images/sleeping_kitten.jpg
```

If you want to include literal `[`, `]` or `` ` `` characters in either the `alt` text or the image label you should escape each such character with a backslash.

Also, the link URL should not have any URL-unsafe characters like `<`, `>` or `` ` `` - if required, they should be escaped, for example, to `&lt;`, `&gt;` or `&#96` respectively.

Inline images

The inline-image syntax looks like this:

```
![Alt text](/path/to/img.jpg)

![Alt text](/path/to/img.jpg "Optional title")
```

That is:

- An exclamation mark: `!`;
- followed by a set of square brackets, containing the `alt` attribute text for the image;
- followed by a set of regular parentheses, containing the URL or path to the image, and an optional title attribute enclosed in double or single quotes.

If you want to include literal `[`, `]` or `` ` `` characters in the `alt` text, you should escape each such character with a backslash.

Like in the case of inline link syntax, if your image URL contains the `(` or `)` characters, or if you want to break the URL across multiple lines, you should enclose the URL in <angle brackets>. Also, the link URL should not have any URL-unsafe characters like `<`, `>` or `` ` `` - if required, they should be escaped, for example, to `&lt;`, `&gt;` or `&#96` respectively.

Miscellaneous

Automatic links

Complete URLs of the form `scheme://path` are automatically converted to links. For example:

```
See http://example.net/page.html for more details.
```

becomes, in HTML output:

```
See <a href="http://example.net/page.html">http://example.net/page.
```

Any trailing punctuation at the end of URLs are not considered as part of the URL. This helps in correctly recognizing URLs immediately followed by punctuation, like:

```
When you go to http://example.net/, keep the other
website (http://example.com/) in mind.
```

However, if you want to include any trailing punctuation in the URL, you should enclose the URL in <angle brackets>, like this:

```
 _Hot Shots!_ (<http://en.wikipedia.org/wiki/Hot_Shots!>) was hilar
```

If you have a long URL that you'd like to split across multiple lines, then again you should use <angle brackets>, like this:

```
You can read about the book at <http://en.wikipedia.org/wiki/The_
Amazing_Maurice_and_his_Educated_Rodents>.
```

Otherwise, the <angle brackets> are optional for URLs of the form `scheme://path`.

In addition, email addresses enclosed in <angle brackets> will be converted to `mailto:` links. For example:

```
Mail me at <me@example.net>.
```

becomes, in HTML output:

```
Mail me at <a href="mailto:me@example.net">me@example.net</a>.
```

If you have some text that would normally get interpreted as a link, but you don't want it to, you can use backslash-escaping:

```
These shalt not be links: http\://example.net <http\://example.net>
Nor this: \<me@example.net>
```

Automatic escaping for HTML output

The HTML syntax requires that `<`, `>` and `&` characters be escaped as character entities. vfmd takes this requirement into consideration when converting a vfmd text to HTML.

Any `&` characters in the input text will appear as `&amp;` in the HTML output. For example, if the vfmd talks about `AT&T`, the HTML output will have it as `AT&amp;T`.

However, if you use a `&` to specify a character reference in vfmd, it would be left intact. For example, if you write `Copyright &copy; 2013`, the `&` in that text will be left intact in the output HTML.

Any `<` or `>` in the input text will be similarly converted to `&lt;` and `&gt;`, unless they form the beginning and end of a HTML tag. For example, if you write `4 < 5`, it will be converted to `4 &lt; 5`.

However, inside vfmd's code spans and code blocks, the `<` and `>`s are coverted to named entities even if they are part of a HTML tag. Similarly, any `&` characters within code spans and code blocks are converted to `&amp;` even if they are part of a character reference. This makes it easy to quote HTML code within code blocks and code spans.

Backslash escapes

You can use backslash escapes to generate literal characters which would otherwise have special meaning in vfmd's formatting syntax.

For example, if you wanted to surround a word with literal asterisks (instead of emphasising the text), you can use backslashes before the asterisks, like this:

```
\*literal asterisks\*
```

The escaping backslashes will not be part of the output. For the above example, the HTML output will be:

```
<p>*literal asterisks*</p>
```

To keep things straightforward, any backslash that appears before a punctuation character is considered an escaping backslash, and will not be part of the output. This is true even if the punctuation character wouldn't otherwise be part of a vfmd construct.

For example:

```
God is Real \(unless declared Integer\).
```

becomes, in HTML output:

```
<p>God is Real (unless declared Integer).</p>
```

Any backslash that appears before a non-punctuation character is not counted as an escaping backslash, and is treated literally.

For example:

```
Please run C:\system32\regedit.exe
```

becomes, in HTML output:

```
<p>Please run C:\system32\regedit.exe</p>
```

Any backslashes appearing in code spans, code blocks and link URLs are not considered to be escaping backslashes, and are treated as literal backslashes.

Mixing HTML with vfmd

If you are capable of authoring HTML documents, you can use snippets of HTML in your vfmd document to augument the vfmd feature set. That said, if you would like to convert a vfmd source document to any output format other than HTML, the support for using HTML tags in the vfmd source document is implementation-dependant.

vfmd handles different HTML elements differently. In vfmd, HTML elements are seen as belonging to one of the following four groups:

1. **Phrasing HTML elements** are HTML elements that belong to the phrasing content category in HTML5. The elements in this group are: `a`, `abbr`, `area`, `audio`, `b`, `bdi`, `bdo`, `br`, `button`, `canvas`, `cite`, `code`, `data`, `datalist`, `del`, `dfn`, `em`, `embed`, `i`, `iframe`, `img`, `input`, `ins`, `kbd`, `keygen`, `label`, `map`, `mark`, `meter`, `noscript`, `object`, `output`, `progress`, `q`, `ruby`, `s`, `samp`, `select`, `small`, `span`, `strong`, `sub`, `sup`, `textarea`, `time`, `u`, `var`, `video` and `wbr`.

2. **Verbatim HTML starter elements** are HTML elements that are flow content, but not phrasing content, and can in turn contain flow content. The elements in this group are: `address`, `article`, `aside`, `blockquote`, `details`, `dialog`, `div`, `dl`, `fieldset`, `figure`, `footer`, `form`, `header`, `main`, `nav`, `ol`, `section`, `table` and `ul`.

3. **Verbatim HTML container elements** are HTML elements within which vfmd syntax should never be recognized. `pre`, `style`

and `script` elements fall in this group.

4. **Other HTML elements** are HTML elements that don't belong to any of the above groups.

Using vfmd along with HTML markup

Phrasing HTML elements can be freely intermingled with vfmd text. Such HTML elements can contain, and be contained in, vfmd markup.

For example:

```
According to the <u>_Special_ Theory of Relativity</u>,
**E** <em>(energy)</em> = **mc<sup>2</sup>**
```

becomes, in HTML output:

```
<p>According to the <u><em>Special</em> Theory of Relativity</u>,
<strong>E</strong> <em>(energy)</em> = <strong>mc<sup>2</sup></stro
```

Other HTML elements can contain vfmd text, but cannot be contained within span-level vfmd markup. Moreover, vfmd paragraphs containing any of the other HTML elements are not wrapped in `p` tags.

The following example shows how vfmd markup can be used within `td` elements of a HTML table:

```
<td>Apply some _stress_</td>
<td>Make it sound **important**</td>
```

The HTML output for the above example shall not have any `p` tags:

```
<td>Apply some <em>stress</em></td>
<td>Make it sound <strong>important</strong></td>
```

Also, if the vfmd paragraph contains any mismatched or misnested HTML tags (i.e. opening tags without correctly-placed closing tags or vice versa), vfmd will not wrap the paragraph content in `p` tags, because doing so might result in invalid HTML output.

Verbatim HTML

Anything within a verbatim HTML container element (i.e. a `pre` , `script` or `style` element) is preserved as-is in the HTML output.

Moreover, any tag (opening, closing or self-closing tag) of a verbatim HTML starter element marks the start of verbatim HTML, and the verbatim-HTML-mode stays on till the next blank line.

So, if you'd like to use vfmd syntax within a verbatim HTML starter element (like `div` or `table`), you can use one or more blank lines to separate the parts that need to be output verbatim, from the parts in which vfmd syntax needs to be processed.

For example, for the input:

```
<table>
  <tr>
    <th>Single *asterisk* or _underscore_</th>
    <th>Double **asterisks** or __underscores__</th>
  </tr>
  <tr>

<td>Apply some _stress_</td>
<td>Make it sound **important**</td>

  </tr>
</table>
```

The corresponding HTML output shall be:

```
<table>
  <tr>
    <th>Single *asterisk* or _underscore_</th>
    <th>Double **asterisks** or __underscores__</th>
  </tr>
  <tr>

<td>Apply some <em>stress</em></td>
<td>Make it sound <strong>important</strong></td>

  </tr>
</table>
```

However, please make sure that the starting line of each snippet of HTML is not indented by more than 3 spaces (if it's indented by 4 or more spaces, it would become a code block).

On the contrary, if you want the whole block of HTML reproduced verbatim in the HTML output, make sure that there are no blank lines in the content of any of the HTML elements.

For example:

```
<table>
  <tr>
    <th>Single *asterisk* or _underscore_</th>
    <th>Double **asterisks** or __underscores__</th>
  </tr>
  <tr>
<td>Apply some _stress_</td>
<td>Make it sound **important**</td>
  </tr>
</table>
```

The text within the `td` tags in the above example shall not be processed as vfmd, and shall be reproduced as-is in the HTML output.

Note that `pre`, `script` or `style` elements can have blank lines enclosed within, and those blank lines don't cause the verbatim-HTML-mode to end.

Please note that the part of the document that is treated as verbatim HTML will be truly output verbatim; No automatic escaping of `<`, `>` or `&` will be done, and any backslashes will be treated as literal backslashes - not as escaping backslashes.

HTML blocks within blockquotes and lists

When including a HTML block within a blockquote, please make sure that the HTML block is completely contained within the blockquote.

For example, the following is wrong, and will result in invalid HTML output:

```
> This is a blockquote.
> Below is a HTML block inside the blockquote.
>
> <div>
> Inside the blockquote and inside the div block

Outside the blockquote
</div>
```

To correctly include the HTML block within the blockquote, please use the `>` prefix for all the lines in the HTML block (or atleast for any line following a blank line in the HTML block).

```
> This is a blockquote.
> Below is a HTML block inside the blockquote.
>
> <div>
> Inside the blockquote and inside the div block
>
> Still inside the blockquote
> </div>
```

Similarly, when including a HTML block within a list, please make sure that the HTML block is completely contained within a list item.

For example, the following is wrong, and will result in invalid HTML output:

```
 1. This is a list.
    Below is a HTML block inside the list.

    <div>
    Inside the list and inside the div block

Outside the list
</div>
```

Instead, please indent all the lines of the HTML block (or atleast the lines following a blank line) to line up with the list.

```
 1. This is a list.
    Below is a HTML block inside the list.

    <div>
    Inside the list and inside the div block

    Still inside the list
    </div>
```

Simarly, when including a HTML comment within a blockquote or a list item, please make sure that the HTML comment is completely contained within the blockquote or the list item.

For example, the HTML comment in the following snippet will not be recognized:

```
* This list item contains the start of a HTML comment
  <!-- HTML comment start
* This list item is not commented
* Because a HTML comment cannot span across
  multiple list items -->
* Last list item
```

If you'd like to comment out a whole blockquote or list, you can consolidate the HTML comment as a standalone HTML block, separated with blank lines, like this:

```
* This list item is followed by a HTML comment

<!-- HTML comment start
* This list item is commented
* This HTML comment is consolidated as
```

```
  a separate HTML block -->

* Another list
```

When using verbatim HTML within a blockquote or a list item, the verbatim-HTML-mode comes to an end at the end of the blockquote or the list item, even if there is no blank line.

For example:

```
* This list item contains a div tag.
  <div>
     Anything following a div tag
     would be interpreted as verbatim HTML.
  </div>
  Markdown sytax **is not** recognized here.
* But this is the next list item, and so
  Markdown sytax **is** very much recognized here.
```

Matching open and close HTML tags

When you use HTML tags within the vfmd document, please take care to correctly match up the open tags with the close tags. Any mismatched or misnested HTML tags will remain mismatched or misnested in the output HTML as well. Some implementations might clean up the invalid HTML, but then their output might not really match what you intended in your input. The best way to be clear about your intent is to provide correctly matching HTML tags in the vfmd source document.

In addition, span-level vfmd markup will not be recognized across mismatching or misnested HTML tags.

For example, the asterisks in the following vfmd input are not interpreted to mean strong emphasis:

```
Cannot span **across <b><i>misnested</b></i> HTML tags** _at all_
```

For them to be recognized as vfmd markup, the HTML tags should be fixed to nest correctly:

```
Can span **across <b><i>correctly nested</i></b> HTML tags** _anyti
```

So, when mixing HTML with vfmd, please ensure that the open HTML tags and the close HTML tags you insert match up correctly.

License

This document is derived from the original Markdown syntax guide, and is published under the same license.
