PHP Markdown Extra

PHP Markdown Extra is a special version of PHP Markdown implementing some features currently not available with the plain Markdown syntax. You can download PHP Markdown Extra from the PHP Markdown home page.

This document explains the changes and additions to the Markdown syntax implemented by PHP Markdown Extra. You should already be familiar with original Markdown syntax documentation before reading this document.

Inline HTML

With Markdown, you can insert HTML right in the middle of your text. This is pretty useful when you need some features not provided by the Markdown syntax but which are easy to do with HTML.

But Markdown has a serious limitation when it comes to block elements. From the Markdown syntax documentation:

> Block-level HTML elements ” e.g. `<div>`, `<table>`, `<pre>`, `<p>`, etc. ” must be separated from surrounding content by blank lines, and the start and end tags of the block should not be indented with tabs or spaces.

These restrictions have been lifted in PHP Markdown Extra, and replaced by these less restrictive two:

1. The opening tag of a block element must not be indented by more than three spaces. Any tag indented more than that will be treated as a code block according to standard Markdown rules.
2. When the block element is found inside a list, all its content should be indented with the same amount of space as the list item is indented. (More indentation won't do any harm as long as the first opening tag is not indented too much and then become a code block -- see first rule.)

Markdown Inside HTML Blocks

Previously in Markdown, you couldn't wrap Markdown-formatted content inside a `<div>` element. This is because `<div>` is a block element and plain Markdown does not format the content of such.

PHP Markdown Extra gives you a way to put Markdown-formatted text inside any block-level tag. You do this by adding a `markdown` attribute to the tag with the value `1` -- which gives `markdown="1"` -- like this:

```
<div markdown="1">
This is *true* markdown text.
</div>
```

The `markdown="1"` attribute will be stripped and `<div>`'s content will be converted from Markdown to HTML. The end result will look like this:

```
<div>

<p>This is <em>true</em> markdown text.</p>

</div>
```

PHP Markdown Extra is smart enough to apply the correct formatting depending on the block element you put the `markdown` attribute on. If you apply the `markdown` attribute to a `<p>` tag for instance, it will only produce span-level elements inside -- it won't allow lists, blockquotes, code blocks.

But these are some cases where this is ambiguous, like this one for instance:

```
<table>
<tr>
<td markdown="1">This is *true* markdown text.</td>
</tr>
</table>
```

A table cell can contain both span and block elements. In cases like this one, PHP Markdown Extra will only apply span-level rules. If you wish to enable block constructs, simply write `markdown="block"` instead.

Special Attributes

With PHP Markdown Extra, you can set the id and class attribute on certain elements using an attribute block. For instance, put the desired id prefixed by a hash inside curly brackets after the header at the end of the line, like this:

```
Header 1            {#header1}
========

## Header 2 ##      {#header2}
```

Then you can create links to different parts of the same document like this:

```
[Link back to header 1](#header1)
```

To add a class name, which can be used as a hook for a style sheet, use a dot like this:

```
## The Site ##    {.main}
```

The id and multiple class names can be combined by putting them all into the same special attribute block:

```
## The Site ##    {.main .shine #the-site}
```

Special attribute blocks can be used only with headers fenced code blocks.

Fenced Code Blocks

Version 1.2 of PHP Markdown Extra introduced a syntax code block without indentation. Fenced code blocks are like Markdown's regular code blocks, except that they're not indented and instead rely on a start and end fence lines to delimit the code block. The code block start with a line containing three or more tilde ~ characters, and ends with the first line with the same number of tilde ~. For instance:

```
This is a paragraph introducing:

~~~~~~~~~~~~~~~~~~~~~
a one-line code block
~~~~~~~~~~~~~~~~~~~~~
```

Contrary to their indented counterparts, fenced code blocks can begin and end with blank lines:

```
~~~

blank line before
blank line after

~~~
```

Indented code blocks cannot be used immediately following a list because the list indent takes precedence; fenced code block have no such limitation:

```
1.  List item

    Not an indented code block, but a second paragraph
    in the list item

~~~~
This is a code block, fenced-style
~~~~
```

Fenced code blocks are also ideal if you need to paste some code in an editor which doesn't have a command for increasing the indent of a block of text, such as a text box in your web browser.

You can specify a class name that will apply to a code block. This is useful if you want to style differently code blocks depending on the language. Or you could also use it to tell a syntax highlighter what syntax to use.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~ .html
<p>paragraph <b>emphasis</b>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The class name is placed at the end of the first fence. It can be preceded by a dot, but this is not a requirement. You can also use a special attribute block:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~ {.html #example-1}
<p>paragraph <b>emphasis</b>
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~

In the HTML output, code block attributes will be applied on the `code` element; if you want to see them on the `pre` element instead, set the configuration constant `MARKDOWN_CODE_ATTR_ON_PRE` at the beginning of the file to `true`.

Tables

PHP Markdown Extra has its own syntax for simple tables. A "simple" table looks like this:

```
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

First line contains column headers; second line contains a mandatory separator line between the headers and the content; each following line is a row in the table. Columns are always separated by the pipe (`|`) character. Once converted to HTML, the result is like this:

```
<table>
<thead>
<tr>
  <th>First Header</th>
  <th>Second Header</th>
</tr>
</thead>
<tbody>
<tr>
  <td>Content Cell</td>
  <td>Content Cell</td>
</tr>
<tr>
  <td>Content Cell</td>
  <td>Content Cell</td>
</tr>
</tbody>
</table>
```

If you wish, you can add a leading and tailing pipe to each line of the table. Use the form that you like. As an illustration, this will give the same result as above:

```
| First Header  | Second Header |
| ------------- | ------------- |
| Content Cell  | Content Cell  |
| Content Cell  | Content Cell  |
```

Note: A table need *at least* one pipe on each line for PHP Markdown Extra to parse it correctly. This means that the only way to create a one-column table is to add a leading or a tailing pipe, or both of them, to each line.

You can specify alignment for each column by adding colons to separator lines. A colon at the left of the separator line will make the column left-aligned; a colon on the right of the line will make the column right-aligned; colons at both side means the column is center-aligned.

```
| Item      | Value |
| --------- | -----:|
| Computer  | $1600 |
| Phone     |   $12 |
| Pipe      |    $1 |
```

The `align` HTML attribute is applied to each cell of the concerned column.

You can apply span-level formatting to the content of each cell using regular Markdown syntax:

```
| Function name | Description                    |
| ------------- | ------------------------------ |
| `help()`      | Display the help window.       |
| `destroy()`   | **Destroy your computer!**     |
```

Definition Lists

PHP Markdown Extra implements definition lists. Definition lists are made of terms and definitions of these terms, much like in a dictionary.

A simple definition list in PHP Markdown Extra is made of a single-line term followed by a colon and the definition for that term.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
Apple
:   Pomaceous fruit of plants of the genus Malus in
    the family Rosaceae.

Orange
:   The fruit of an evergreen tree of the genus Citrus.
```

Terms must be separated from the previous definition by a blank line. Definitions can span on multiple lines, in which case they should be indented. But you don't really have to: if you want to be lazy, you could forget to indent a definition that span on multiple lines and it will still work:

```
Apple
:   Pomaceous fruit of plants of the genus Malus in
the family Rosaceae.

Orange
:   The fruit of an evergreen tree of the genus Citrus.
```

Each of the preceding definition lists will give the same HTML result:

```
<dl>
<dt>Apple</dt>
<dd>Pomaceous fruit of plants of the genus Malus in
the family Rosaceae.</dd>

<dt>Orange</dt>
<dd>The fruit of an evergreen tree of the genus Citrus.</dd>
</dl>
```

Colons as definition markers typically start at the left margin, but may be indented by up to three spaces. Definition markers must be followed by one or more spaces or a tab.
Definition lists can have more than one definition associated with one term:

```
Apple
:   Pomaceous fruit of plants of the genus Malus in
    the family Rosaceae.
:   An American computer company.

Orange
:   The fruit of an evergreen tree of the genus Citrus.
```

You can also associate more than one term to a definition:

```
Term 1
Term 2
:   Definition a

Term 3
:   Definition b
```

If a definition is preceded by a blank line, PHP Markdown Extra will wrap the definition in <p> tags in the HTML output. For example, this:

```
Apple

:   Pomaceous fruit of plants of the genus Malus in
    the family Rosaceae.

Orange

:    The fruit of an evergreen tree of the genus Citrus.
```

will turn into this:

```
<dl>
<dt>Apple</dt>
```

```
<dd>
<p>Pomaceous fruit of plants of the genus Malus in
the family Rosaceae.</p>
</dd>

<dt>Orange</dt>
<dd>
<p>The fruit of an evergreen tree of the genus Citrus.</p>
</dd>
</dl>
```

And just like regular list items, definitions can contain multiple paragraphs, and include other block-level elements such as blockquotes, code blocks, lists, and even other definition lists.

```
Term 1

:   This is a definition with two paragraphs. Lorem ipsum
    dolor sit amet, consectetuer adipiscing elit. Aliquam
    hendrerit mi posuere lectus.

    Vestibulum enim wisi, viverra nec, fringilla in, laoreet
    vitae, risus.

:   Second definition for term 1, also wrapped in a paragraph
    because of the blank line preceding it.

Term 2

:   This definition has a code block, a blockquote and a list.

        code block.

    > block quote
    > on two lines.

    1.  first list item
    2.  second list item
```

Footnotes

Footnotes work mostly like reference-style links. A footnote is made of two things: a marker in the text that will become a superscript number; a footnote definition that will be placed in a list of footnotes at the end of the document. A footnote looks like this:

```
That's some text with a footnote.[^1]

[^1]: And that's the footnote.
```

Footnote definitions can be found anywhere in the document, but footnotes will always be listed in the order they are linked to in the text. Note that you cannot make two links to the same footnotes: if you try, the second footnote reference will be left as plain text.

Each footnote must have a distinct name. That name will be used to link footnote references to footnote definitions, but has no effect on the numbering of the footnotes. Names can contain any character valid within an `id` attribute in HTML.

Footnotes can contain block-level elements, which means that you can put multiple paragraphs, lists, blockquotes and so on in a footnote. It works the same as for list items: just indent the following paragraphs by four spaces in the footnote definition:

```
That's some text with a footnote.[^1]

[^1]: And that's the footnote.

    That's the second paragraph.
```

If you want things to align better, you can leave the first line of the footnote empty and put your first paragraph just below:

```
[^1]:
    And that's the footnote.
    That's the second paragraph.
```

Output

It's probably true that a single footnote markup cannot satisfy everyone. A future version may provide a programming interface to allow different markup to be generated. But for today, the output follows what can be seen on Daring Fireball, with slight modifications. Here is the default output from the first sample above:

```
<p>That's some text with a footnote.
   <sup id="fnref:1"><a href="#fn:1" rel="footnote">1</a></sup></p>

<div class="footnotes">
<hr />
<ol>

<li id="fn:1">
<p>And that's the footnote.
   <a href="#fnref:1" rev="footnote">↩</a></p>
</li>

</ol>
</div>
```

A little cryptic, but in a browser it will look like this:

That's some text with a footnote.[1]

---

1. And that's the footnote. □

The `rel` and `rev` attributes on the links express the relation they have with the elements they link to. Reference links lead to footnotes (hence `rel="footnote"`) and back-links comes from footnotes (hence `rev="footnote"`). They can be used to style the elements with CSS rules such as:

```
a[rel="footnote"]
a[rev="footnote"]
```

You can also customize the `class` and `title` attributes for footnote links and back-links. There are four configurable settings to that effect at the start of the file. Within these attributes, any occurrence of `%%` will be replaced by the current footnote number. For instance, you could use:

```
define( 'MARKDOWN_FN_LINK_TITLE', "Go to footnote %%." );
```

Abbreviations

PHP Markdown Extra adds supports for abbreviations (HTML tag `<abbr>`). How it works is pretty simple: create an abbreviation definition like this:

```
*[HTML]: Hyper Text Markup Language
*[W3C]:  World Wide Web Consortium
```

then, elsewhere in the document, write text such as:

```
The HTML specification
is maintained by the W3C.
```

and any instance of those words in the text will become:

```
The <abbr title="Hyper Text Markup Language">HTML</abbr> specification
is maintained by the <abbr title="World Wide Web Consortium">W3C</abbr>.
```

Abbreviations are case-sensitive, and will span on multiple words when defined as such. An abbreviation may also have an empty definition, in which case `<abbr>` tags will be added in the text but the `title` attribute will be omitted.

```
Operation Tigra Genesis is going well.

*[Tigra Genesis]:
```

Abbreviation definitions can be anywhere in the document. They are stripped from the final document.

Emphasis

Rules for emphasis have slightly changed from the original Markdown syntax. With PHP Markdown Extra, underscores in the middle of a word are now treated as literal characters. Underscore emphasis only works for whole words. If you need to emphasize only some part of a word, it is still possible by using

asterisks as emphasis markers.
For example, with this:

```
Please open the folder "secret_magic_box".
```

PHP Markdown Extra won't convert underscores to emphasis because they are in the middle of the word. The HTML result from PHP Markdown Extra looks like this:

```
<p>Please open the folder "secret_magic_box".</p>
```

Emphasis with underscore still works as long as you emphasize whole words like this:

```
I like it when you say _you love me_.
```

The same apply for strong emphasis: with PHP Markdown Extra, you can no longer set strong emphasis in the middle of a word using underscores, you must do so using asterisks as emphasis markers.

Backslash Escapes

PHP Markdown Extra adds the colon (`:`) and the pipe (`|`) to the list of characters you can escape using a backslash. With this you can prevent them from triggering a definition list or a table.

Thanks

Many ideas implemented here have been discussed before on the Markdown discussion list. I want to thank everyone who have participated in these discussions and drafted solutions and improvements to the Markdown syntax.