# Markdown Types

- Markdown Syntax
- MarkdownExtra Syntax

# Markdown: Syntax



---

## Overview

### Philosophy

Markdown is intended to be as easy-to-read and easy-to-write as is feasible.

Readability, however, is emphasized above all else. A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. While Markdown's syntax has been influenced by several existing text-to-HTML filters -- including Setext, atx, Textile, reStructuredText, Grutatext, and EtText -- the single biggest source of inspiration for Markdown's syntax is the format of plain text email.

To this end, Markdown's syntax is comprised entirely of punctuation characters, which punctuation characters have been carefully chosen so as to look like what they mean. E.g., asterisks around a word actually look like *emphasis*. Markdown lists look like, well, lists. Even blockquotes look like quoted passages of text, assuming you've ever used email.

### Inline HTML

Markdown's syntax is intended for one purpose: to be used as a format for *writing* for the web.

Markdown is not a replacement for HTML, or even close to it. Its syntax is very small, corresponding only to a very small subset of HTML tags. The idea is *not* to create a syntax that makes it easier to insert HTML tags. In my opinion, HTML tags are already easy to insert. The idea for Markdown is to make it easy to read, write, and edit prose. HTML is a *publishing* format; Markdown is a *writing* format. Thus, Markdown's formatting syntax only addresses issues that can be conveyed in plain text.

For any markup that is not covered by Markdown's syntax, you simply use HTML itself. There's no need to preface it or delimit it to indicate that you're switching from Markdown to HTML; you just use the tags.

The only restrictions are that 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. Markdown is smart enough not to add extra (unwanted) <p> tags around HTML block-level tags.

For example, to add an HTML table to a Markdown article:

```
This is a regular paragraph.

<table>
    <tr>
        <td>Foo</td>
    </tr>
</table>

This is another regular paragraph.
```

Note that Markdown formatting syntax is not processed within block-level HTML tags. E.g., you can't use Markdown-style `*emphasis*` inside an HTML block.

Span-level HTML tags -- e.g. `<span>`, `<cite>`, or `<del>` -- can be used anywhere in a Markdown paragraph, list item, or header. If you want, you can even

use HTML tags instead of Markdown formatting; e.g. if you'd prefer to use HTML <a> or <img> tags instead of Markdown's link or image syntax, go right ahead.

Unlike block-level HTML tags, Markdown syntax *is* processed within span-level tags.

### Automatic Escaping for Special Characters

In HTML, there are two characters that demand special treatment: < and &. Left angle brackets are used to start tags; ampersands are used to denote HTML entities. If you want to use them as literal characters, you must escape them as entities, e.g. `<`, and `&`.

Ampersands in particular are bedeviling for web writers. If you want to write about 'AT&T', you need to write 'AT`&`T'. You even need to escape ampersands within URLs. Thus, if you want to link to:

```
http://images.google.com/images?num=30&q=larry+bird
```

you need to encode the URL as:

```
http://images.google.com/images?num=30&amp;q=larry+bird
```

in your anchor tag `href` attribute. Needless to say, this is easy to forget, and is probably the single most common source of HTML validation errors in otherwise well-marked-up web sites.

Markdown allows you to use these characters naturally, taking care of all the necessary escaping for you. If you use an ampersand as part of an HTML entity, it remains unchanged; otherwise it will be translated into `&`.

So, if you want to include a copyright symbol in your article, you can write:

```
&copy;
```

and Markdown will leave it alone. But if you write:

```
AT&T
```

Markdown will translate it to:

```
AT&amp;T
```

Similarly, because Markdown supports inline HTML, if you use angle brackets as delimiters for HTML tags, Markdown will treat them as such. But if you write:

```
4 < 5
```

Markdown will translate it to:

```
4 &lt; 5
```

However, inside Markdown code spans and blocks, angle brackets and ampersands are *always* encoded automatically. This makes it easy to use Markdown to write about HTML code. (As opposed to raw HTML, which is a terrible format for writing about HTML syntax, because every single < and & in your example code needs to be escaped.)

---

## Block Elements

### Paragraphs and Line Breaks

A paragraph is simply 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 intended with spaces or tabs.

The implication of the "one or more consecutive lines of text" rule is that Markdown supports "hard-wrapped" text paragraphs. This differs significantly from most other text-to-HTML formatters (including Movable Type's "Convert Line Breaks" option) which translate every line break character in a paragraph into a `<br />` tag.

When you *do* want to insert a `<br />` break tag using Markdown, you end a line with two or more spaces, then type return.

Yes, this takes a tad more effort to create a `<br />`, but a simplistic "every line break is a `<br />`" rule wouldn't work for Markdown. Markdown's email-style blockquoting and multi-paragraph list items work best -- and look better -- when you format them with hard breaks.

### Headers

Markdown supports two styles of headers, Setext and atx.

Setext-style headers are "underlined" using equal signs (for first-level headers) and dashes (for second-level headers). For example:

```
This is an H1
=============

This is an H2
-------------
```

Any number of underlining ='s or `-`'s will work.

Atx-style headers use 1-6 hash characters at the start of the line, corresponding to header levels 1-6. For example:

```
# This is an H1

## This is an H2

###### This is an H6
```

Optionally, you may "close" atx-style headers. This is purely cosmetic -- you can use this if you think it looks better. The closing hashes don't even need to match the number of hashes used to open the header. (The number of opening hashes determines the header level.) :

```
# This is an H1 #

## This is an H2 ##

### This is an H3 ######
```

### Blockquotes

Markdown uses email-style > characters for blockquoting. If you're familiar with quoting passages of text in an email message, then you know how to create a blockquote in Markdown. It looks best if you hard wrap the text and put a > before every line:

```
> This is a blockquote 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.
>
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
> id sem consectetuer libero luctus adipiscing.
```

Markdown allows you to be lazy and only put the > before the first line of a hard-wrapped paragraph:

```
> This is a blockquote 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.

> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
id sem consectetuer libero luctus adipiscing.
```

Blockquotes can be nested (i.e. a blockquote-in-a-blockquote) by adding additional levels of >:

```
> This is the first level of quoting.
>
> > This is nested blockquote.
>
> Back to the first level.
```

Blockquotes can contain other Markdown elements, including headers, lists, and code blocks:

```
> ## This is a header.
>
> 1.   This is the first list item.
> 2.   This is the second list item.
>
> Here's some example code:
>
>     return shell_exec("echo $input | $markdown_script");
```

Any decent text editor should make email-style quoting easy. For example, with BBEdit, you can make a selection and choose Increase Quote Level from the Text menu.

### Lists

Markdown supports ordered (numbered) and unordered (bulleted) lists.

Unordered lists use asterisks, pluses, and hyphens -- interchangably -- as list markers:

```
*   Red
*   Green
*   Blue
```

is equivalent to:

```
+   Red
+   Green
+   Blue
```

and:

```
-   Red
-   Green
-   Blue
```

Ordered lists use numbers followed by periods:

```
1.  Bird
2.  McHale
3.  Parish
```

It's important to note that the actual numbers you use to mark the list have no effect on the HTML output Markdown produces. The HTML Markdown produces from the above list is:

```
<ol>
<li>Bird</li>
<li>McHale</li>
<li>Parish</li>
</ol>
```

If you instead wrote the list in Markdown like this:

```
1.  Bird
1.  McHale
1.  Parish
```

or even:

```
3. Bird
1. McHale
8. Parish
```

you'd get the exact same HTML output. The point is, if you want to, you can use ordinal numbers in your ordered Markdown lists, so that the numbers in your source match the numbers in your published HTML. But if you want to be lazy, you don't have to.
If you do use lazy list numbering, however, you should still start the list with the number 1. At some point in the future, Markdown may support starting ordered lists at an arbitrary number.
List markers typically start at the left margin, but may be indented by up to three spaces. List markers must be followed by one or more spaces or a tab.
To make lists look nice, you can wrap items with hanging indents:

```
*   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,
    viverra nec, fringilla in, laoreet vitae, risus.
*   Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
    Suspendisse id sem consectetuer libero luctus adipiscing.
```

But if you want to be lazy, you don't have to:

```
*   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,
viverra nec, fringilla in, laoreet vitae, risus.
*   Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
Suspendisse id sem consectetuer libero luctus adipiscing.
```

If list items are separated by blank lines, Markdown will wrap the items in <p> tags in the HTML output. For example, this input:

```
*   Bird
*   Magic
```

will turn into:

```
<ul>
<li>Bird</li>
```

```
<li>Magic</li>
</ul>
```

But this:

```
*   Bird

*   Magic
```

will turn into:

```
<ul>
<li><p>Bird</p></li>
<li><p>Magic</p></li>
</ul>
```

List items may consist of multiple paragraphs. Each subsequent paragraph in a list item must be intended by either 4 spaces or one tab:

```
1.  This is a list item 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. Donec sit amet nisl. Aliquam semper ipsum
    sit amet velit.

2.  Suspendisse id sem consectetuer libero luctus adipiscing.
```

It looks nice if you indent every line of the subsequent paragraphs, but here again, Markdown will allow you to be lazy:

```
*   This is a list item with two paragraphs.

    This is the second paragraph in the list item. You're
only required to indent the first line. Lorem ipsum dolor
sit amet, consectetuer adipiscing elit.

*   Another item in the same list.
```

To put a blockquote within a list item, the blockquote's > delimiters need to be indented:

```
*   A list item with a blockquote:

    > This is a blockquote
    > inside a list item.
```

To put a code block within a list item, the code block needs to be indented *twice* -- 8 spaces or two tabs:

```
*   A list item with a code block:

        <code goes here>
```

It's worth noting that it's possible to trigger an ordered list by accident, by writing something like this:

```
1986. What a great season.
```

In other words, a *number-period-space* sequence at the beginning of a line. To avoid this, you can backslash-escape the period:

```
1986\. What a great season.
```

## Code Blocks

Pre-formatted code blocks are used for writing about programming or markup source code. Rather than forming normal paragraphs, the lines of a code block are interpreted literally. Markdown wraps a code block in both `<pre>` and `<code>` tags.
To produce a code block in Markdown, simply indent every line of the block by at least 4 spaces or 1 tab. For example, given this input:

```
This is a normal paragraph:

    This is a code block.
```

Markdown will generate:

```
<p>This is a normal paragraph:</p>

<pre><code>This is a code block.
</code></pre>
```

One level of indentation -- 4 spaces or 1 tab -- is removed from each line of the code block. For example, this:

```
Here is an example of AppleScript:

    tell application "Foo"
        beep
    end tell
```

will turn into:

```
<p>Here is an example of AppleScript:</p>

<pre><code>tell application "Foo"
    beep
end tell
</code></pre>
```

A code block continues until it reaches a line that is not indented (or the end of the article).
Within a code block, ampersands (&) and angle brackets (< and >) are automatically converted into HTML entities. This makes it very easy to include example HTML source code using Markdown -- just paste it and indent it, and Markdown will handle the hassle of encoding the ampersands and angle brackets. For example, this:

```
    <div class="footer">
        &copy; 2004 Foo Corporation
    </div>
```

will turn into:

```
<pre><code>&lt;div class="footer"&gt;
    &amp;copy; 2004 Foo Corporation
&lt;/div&gt;
</code></pre>
```

Regular Markdown syntax is not processed within code blocks. E.g., asterisks are just literal asterisks within a code block. This means it's also easy to use Markdown to write about Markdown's own syntax.

### Horizontal Rules

You can produce a horizontal rule tag (`<hr />`) by placing three or more hyphens, asterisks, or underscores on a line by themselves. If you wish, you may use spaces between the hyphens or asterisks. Each of the following lines will produce a horizontal rule:

```
* * *

***

*****

- - -

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

_ _ _
```

---

## Span Elements

### Links

Markdown supports two style of links: *inline* and *reference*.
In both styles, the link text is delimited by [square brackets].
To create an inline link, use a set of regular parentheses immediately after the link text's closing square bracket. Inside the parentheses, 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:

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

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

If you're referring to a local resource on the same server, you can use relative paths:

```
See my [About](/about/) page for details.
```

Reference-style links use a second set of square brackets, inside which you place a label of your choosing to identify the link:

```
This is [an example][id] reference-style link.
```

You can optionally use a space to separate the sets of brackets:

```
This is [an example] [id] reference-style link.
```

Then, anywhere in the document, you define your link label like this, on a line by itself:

```
[id]: http://example.com/  "Optional Title Here"
```

That is:
- Square brackets containing the link identifier (optionally indented from the left margin using 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 or single quotes.

The link URL may, optionally, be surrounded by angle brackets:

```
[id]: <http://example.com/>  "Optional Title Here"
```

You can put the title attribute on the next line and use extra spaces or tabs for padding, which tends to look better with longer URLs:

```
[id]: http://example.com/longish/path/to/resource/here
    "Optional Title Here"
```

Link definitions are only used for creating links during Markdown processing, and are stripped from your document in the HTML output.
Link definition names may constist of letters, numbers, spaces, and punctuation -- but they are *not* case sensitive. E.g. these two links:

```
[link text][a]
[link text][A]
```

are equivalent.
The *implicit link name* shortcut allows you to omit the name of the link, in which case the link text itself is used as the name. Just use an empty set of square brackets -- e.g., to link the word "Google" to the google.com web site, you could simply write:

```
[Google][]
```

And then define the link:

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

Because link names may contain spaces, this shortcut even works for multiple words in the link text:

```
Visit [Daring Fireball][] for more information.
```

And then define the link:

```
[Daring Fireball]: http://daringfireball.net/
```

Link definitions can be placed anywhere in your Markdown document. I tend to put them immediately after each paragraph in which they're used, but if you want, you can put them all at the end of your document, sort of like footnotes.
Here's an example of reference links in action:

```
I get 10 times more traffic from [Google] [1] than from
[Yahoo] [2] or [MSN] [3].

  [1]: http://google.com/        "Google"
  [2]: http://search.yahoo.com/  "Yahoo Search"
  [3]: http://search.msn.com/    "MSN Search"
```

Using the implicit link name shortcut, you could instead write:

```
I get 10 times more traffic from [Google][] than from
[Yahoo][] or [MSN][].

  [google]: http://google.com/        "Google"
  [yahoo]:  http://search.yahoo.com/  "Yahoo Search"
  [msn]:    http://search.msn.com/    "MSN Search"
```

Both of the above examples will produce the following HTML output:

```
<p>I get 10 times more traffic from <a href="http://google.com/"
title="Google">Google</a> than from
<a href="http://search.yahoo.com/" title="Yahoo Search">Yahoo</a>
or <a href="http://search.msn.com/" title="MSN Search">MSN</a>.</p>
```

For comparison, here is the same paragraph written using Markdown's inline link style:

```
I get 10 times more traffic from [Google](http://google.com/ "Google")
than from [Yahoo](http://search.yahoo.com/ "Yahoo Search") or
[MSN](http://search.msn.com/ "MSN Search").
```

The point of reference-style links is not that they're easier to write. The point is that with reference-style links, your document source is vastly more readable. Compare the above examples: using reference-style links, the paragraph itself is only 81 characters long; with inline-style links, it's 176 characters; and as raw HTML, it's 234 characters. In the raw HTML, there's more markup than there is text.
With Markdown's reference-style links, a source document much more closely resembles the final output, as rendered in a browser. By allowing you to move the markup-related metadata out of the paragraph, you can add links without interrupting the narrative flow of your prose.

## Emphasis

Markdown treats asterisks (*) and underscores (_) as indicators of emphasis. Text wrapped with one * or _ will be wrapped with an HTML  tag; double *'s or _'s will be wrapped with an HTML `<strong>` tag. E.g., this input:

```
*single asterisks*

_single underscores_

**double asterisks**

__double underscores__
```

will produce:

```
<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 can be used in the middle of a word:

```
un*fucking*believable
```

But if you surround an * or _ with spaces, it'll be treated as a literal asterisk or underscore.
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:

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

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

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

which will produce this:

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

The backtick delimiters surrounding a code span may include spaces -- one after the opening, one before the closing. This allows you to place literal backtick characters at the beginning or end of a code span:

```
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></p>
```

With a code span, ampersands and angle brackets are encoded as HTML entities automatically, which makes it easy to include example HTML tags. Markdown will turn this:

```
Please don't use any `<blink>` tags.
```

into:

```
<p>Please don't use any <code>&lt;blink&gt;</code> tags.</p>
```

You can write this:

```
`&#8212;` is the decimal-encoded equivalent of `&mdash;`.
```

to produce:

```
<p><code>&amp;#8212;</code> is the decimal-encoded
equivalent of <code>&amp;mdash;</code>.</p>
```

### Images

Admittedly, it's fairly difficult to devise a "natural" syntax for placing images into a plain text document format.
Markdown uses an image syntax that is intended to resemble the syntax for links, allowing for two styles: *inline* and *reference*.
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 parentheses, containing the URL or path to the image, and an optional `title` attribute enclosed in double or single quotes.

Reference-style image syntax looks like this:

```
![Alt text][id]
```

Where "id" is the name of a defined image reference. Image references are defined using syntax identical to link references:

```
[id]: url/to/image  "Optional title attribute"
```

As of this writing, Markdown has no syntax for specifying the dimensions of an image; if this is important to you, you can simply use regular HTML `<img>` tags.

---

## Miscellaneous

### Automatic Links

Markdown supports a shortcut style for creating "automatic" links for URLs and email addresses: simply surround the URL or email address with angle brackets. What this means is that if you want to show the actual text of a URL or email address, and also have it be a clickable link, you can do this:

```
<http://example.com/>
```

Markdown will turn this into:

```
<a href="http://example.com/">http://example.com/</a>
```

Automatic links for email addresses work similarly, except that Markdown will also perform a bit of randomized decimal and hex entity-encoding to help obscure your address from address-harvesting spambots. For example, Markdown will turn this:

```
<address@example.com>
```

into something like this:

```
<a href="&#x6D;&#x61;i&#x6C;&#x74;&#x6F;:&#x61;&#x64;&#x64;&#x72;&#x65;
&#115;&#115;&#64;&#101;&#120;&#x61;&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;
&#109;">&#x61;&#x64;&#x64;&#x72;&#x65;&#115;&#115;&#64;&#101;&#120;&#x61;
&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;&#109;</a>
```

which will render in a browser as a clickable link to "address@example.com".
(This sort of entity-encoding trick will indeed fool many, if not most, address-harvesting bots, but it definitely won't fool all of them. It's better than nothing, but an address published in this way will probably eventually start receiving spam.)

### Backslash Escapes

Markdown allows you to use backslash escapes to generate literal characters which would otherwise have special meaning in Markdown's formatting syntax. For example, if you wanted to surround a word with literal asterisks (instead of an HTML `<em>` tag), you can backslashes before the asterisks, like this:

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

Markdown provides backslash escapes for the following characters:

```
\   backslash
`   backtick
*   asterisk
_   underscore
{}  curly braces
[]  square brackets
()  parentheses
#   hash mark
+   plus sign
-   minus sign (hyphen)
.   dot
!   exclamation mark
```

# 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
- Markdown Inside HTML Blocks
- Special Attributes
- Fenced Code Blocks

---

## 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.