How to use Markdown for writing Docs

03/26/2019 • 10 minutes to read • +8

In this article

Docs.microsoft.com articles are written in a lightweight markup language called Markdown, which is both easy to read and easy to learn. Because of this, it has quickly become an industry standard. The docs site uses the Markdig flavor of Markdown.

Markdown basics

Headings

To create a heading, you use a hash mark (#), as follows:

```
# This is heading 1
## This is heading 2
### This is heading 3
#### This is heading 4
```

Headings should be done using atx-style, that is, use 1-6 hash characters (#) at the start of the line to indicate a heading, corresponding to HTML headings levels H1 through H6. Examples of first- through fourth-level headers are used above.

There **must** be only one first-level heading (H1) in your topic, which will be displayed as the on-page title.

If your heading finishes with a `#` character, you need to add an extra `#` character in the end in order for the title to render correctly. For example, `# Async Programming`

```
in F# #.
```

Second-level headings will generate the on-page TOC that appears in the "In this article" section underneath the on-page title.

Bold and italic text

To format text as **bold**, you enclose it in two asterisks:

```
This text is **bold**.
```

To format text as *italic*, you enclose it in a single asterisk:

```
This text is *italic*.
```

To format text as both ***bold and italic***, you enclose it in three asterisks:

```
This is text is both ***bold and italic***.
```

Blockquotes

Blockquotes are created using the > character:

```
> The drought had lasted now for ten million years, and the reign of
the terrible lizards had long since ended. Here on the Equator, in the
continent which would one day be known as Africa, the battle for exis-
tence had reached a new climax of ferocity, and the victor was not yet
in sight. In this barren and desiccated land, only the small or the
swift or the fierce could flourish, or even hope to survive.
```

The preceding example renders as follows:

> The drought had lasted now for ten million years, and the reign of the terrible lizards had long since ended. Here on the Equator, in the continent which would one day be known as Africa, the battle for existence had reached a new climax

> of ferocity, and the victor was not yet in sight. In this barren and desiccated land, only the small or the swift or the fierce could flourish, or even hope to survive.

Lists

Unordered list

To format an unordered/bulleted list, you can use either asterisks or dashes. For example, the following Markdown:

```
- List item 1
- List item 2
- List item 3
```

will be rendered as:

- List item 1
- List item 2
- List item 3

To nest a list within another list, indent the child list items. For example, the following Markdown:

```
- List item 1
  - List item A
  - List item B
- List item 2
```

will be rendered as:

- List item 1
  - List item A
  - List item B
- List item 2

Ordered list

To format an ordered/stepwise list, you use corresponding numbers. For example,

the following Markdown:

```
1. First instruction
1. Second instruction
1. Third instruction
```

will be rendered as:

1. First instruction
2. Second instruction
3. Third instruction

To nest a list within another list, indent the child list items. For example, the following Markdown:

```
1. First instruction
   1. Sub-instruction
   1. Sub-instruction
1. Second instruction
```

will be rendered as:

1. First instruction
    a. Sub-instruction
    b. Sub-instruction
2. Second instruction

Note that we use '1.' for all entries. It makes diffs easier to review when later updates include new steps or remove existing steps.

Tables

Tables are not part of the core Markdown specification, but GFM supports them. You can create tables by using the pipe (|) and hyphen (-) characters. Hyphens create each column's header, while pipes separate each column. Include a blank line before your table so it's rendered correctly.

For example, the following Markdown:

```
| Fun                  | With                  | Tables          |
| :------------------- | --------------------: |:---------------:|
| left-aligned column  | right-aligned column | centered column |
| $100                 | $100                  | $100            |
| $10                  | $10                   | $10             |
| $1                   | $1                    | $1              |
```

Will be rendered as:

Fun	With	Tables
left-aligned column	right-aligned column	centered column
$100	$100	$100
$10	$10	$10
$1	$1	$1

For more information on creating tables, see:

- GitHub's Organizing information with tables.
- The Markdown Tables Generator web app.
- Adam Pritchard's Markdown Cheatsheet.
- Michel Fortin's Markdown Extra.
- Convert HTML tables to Markdown.

Links

The Markdown syntax for an inline link consists of the `[link text]` portion, which is the text that will be hyperlinked, followed by the `(file-name.md)` portion, which is the URL or file name that's being linked to:

```
[link text](file-name.md)
```

For more information on linking, see:

- The Markdown syntax guide for details on Markdown's base linking support.
- The Links section of this guide for details on the additional linking syntax that Markdig provides.

Code snippets

Markdown supports the placement of code snippets both inline in a sentence and as a separate "fenced" block between sentences. For details, see:

- Markdown's native support for code blocks
- GFM support for code fencing and syntax highlighting

Fenced code blocks are an easy way to enable syntax highlighting for your code snippets. The general format for fenced code blocks is:

````
```alias
...
your code goes in here
...
```
````

The alias after the initial three backtick (`) characters defines the syntax highlighting to be used. The following is a list of commonly used programming languages in Docs content and the matching label:

These languages have friendly name support and most have language highlighting.

Name	Markdown Label
.NET Console	dotnetcli
ASP.NET (C#)	aspx-csharp
ASP.NET (VB)	aspx-vb
AzCopy	azcopy
Azure CLI	azurecli
Azure PowerShell	azurepowershell
Bash	bash
C++	cpp
C++/CX	cppcx
C++/WinRT	cppwinrt

Name	Markdown Label
C#	csharp
C# in browser	csharp-interactive
Console	console
CSHTML	cshtml
DAX	dax
Docker	dockerfile
F#	fsharp
Go	go
HTML	html
HTTP	http
Java	java
JavaScript	javascript
JSON	json
Kusto Query Language	kusto
Markdown	md
Objective-C	objc
OData	odata
PHP	php
PowerApps (dot decimal separator)	powerapps-dot
PowerApps (comma decimal separator)	powerapps-comma
PowerShell	powershell

Name	Markdown Label
Python	python
Q#	qsharp
R	r
Ruby	ruby
SQL	sql
Swift	swift
TypeScript	typescript
VB	vb
XAML	xaml
XML	xml

The `csharp-interactive` name specifies the C# language, and the ability to run the samples from the browser. These snippets are compiled and executed in a Docker container, and the results of that program execution are displayed in the user's browser window.

Example: C#

Markdown

````
```csharp
// Hello1.cs
public class Hello1
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, World!");
    }
}
```
````

Render

```
// Hello1.cs
public class Hello1
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, World!");
    }
}
```

Example: SQL

Markdown

````
```sql
CREATE TABLE T1 (
  c1 int PRIMARY KEY,
  c2 varchar(50) SPARSE NULL
);
```
````

Render

```
CREATE TABLE T1 (
  c1 int PRIMARY KEY,
  c2 varchar(50) SPARSE NULL
);
```

Docs custom Markdown extensions

> **Note**
>
> Docs.Microsoft.com (Docs) implements a Markdig Parser for Markdown, which is highly compatible with GitHub Flavored Markdown (GFM). Markdig adds some functionality through Markdown extensions. As such, selected articles from the full OPS Authoring Guide are included in this guide for reference. (For example, see "Markdig and Markdown extensions" and "Code snippets" in the

table of contents.)

Docs articles use GFM for most article formatting, such as paragraphs, links, lists, and headings. For richer formatting, articles can use Markdig features such as:

- Note blocks
- Include files
- Selectors
- Embedded videos
- Code snippets/samples

For the complete list, refer to "Markdig and Markdown extensions" and "Code snippets" in the table of contents.

Note blocks

You can choose from four types of note blocks to draw attention to specific content:

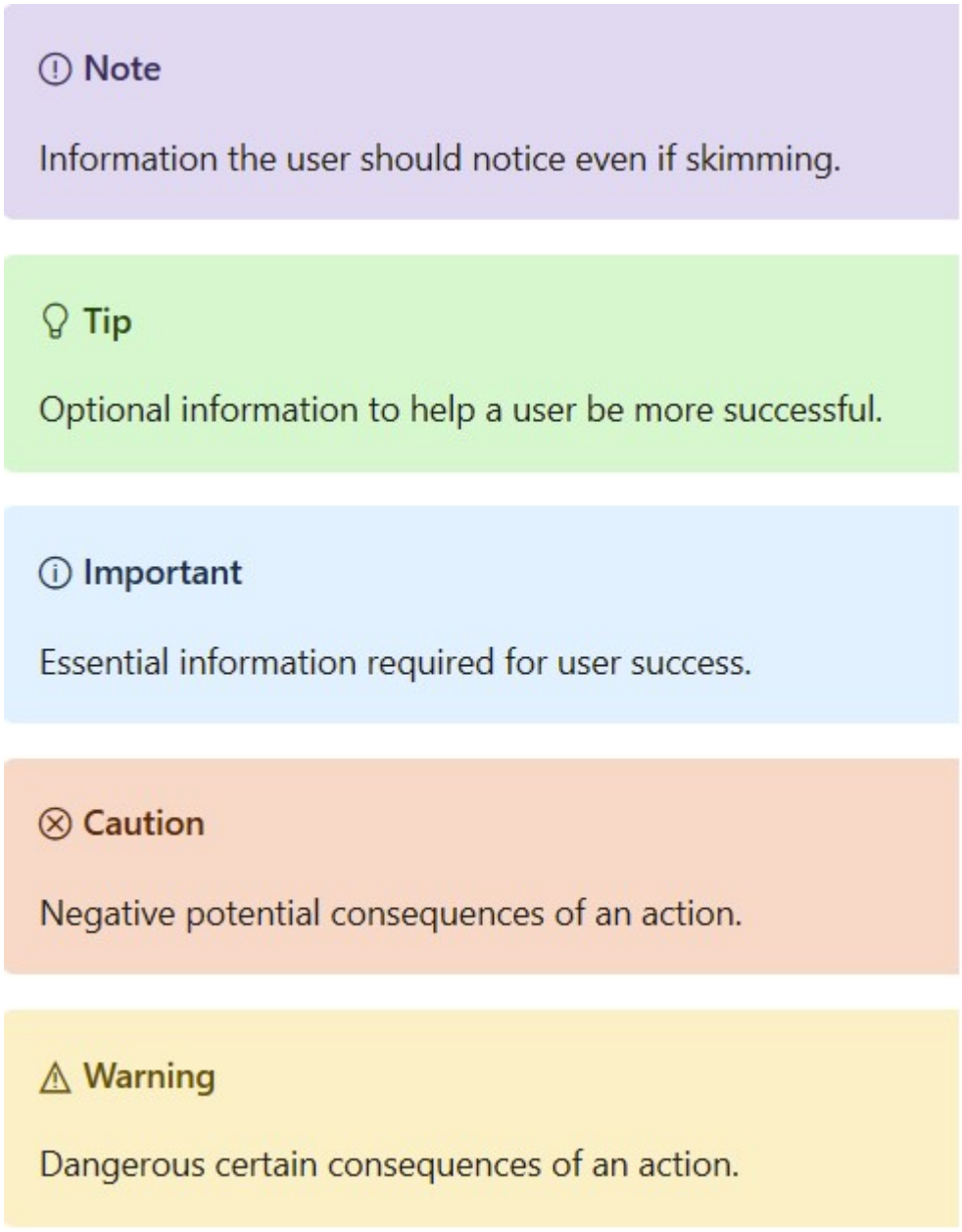
- NOTE
- WARNING
- TIP
- IMPORTANT

In general, note blocks should be used sparingly because they can be disruptive. Although they also support code blocks, images, lists, and links, try to keep your note blocks simple and straightforward.

Examples:

```
> [!NOTE]
> Information the user should notice even if skimming.

> [!TIP]
> Optional information to help a user be more successful.

> [!IMPORTANT]
> Essential information required for user success.

> [!CAUTION]
> Negative potential consequences of an action.

> [!WARNING]
> Dangerous certain consequences of an action.
```

These alerts look like this on docs.microsoft.com:



Include files

When you have reusable text or image files that need to be included in article files, you can use a reference to the "include" file via the Markdig file include feature. This feature instructs OPS to include the file in your article file at build time, making it part of your published article. Three types of include references are available to help you reuse content:

- Inline: Reuse a common text snippet inline with within another sentence.
- Block: Reuse an entire Markdown file as a block, nested within a section of an article.
- Image: This is how standard image inclusion is implemented in Docs.

An inline or block include file is just a simple Markdown (.md) file. It can contain any valid Markdown. All include Markdown files should be placed in a common /includes subdirectory, in the root of the repository. When the article is published, the included file is seamlessly integrated into it.

Here are requirements and considerations for include files:

- Use an include file whenever you need the same text to appear in multiple articles.
- Use a block include reference for significant amounts of content--a paragraph or two, a shared procedure, or a shared section. Do not use them for anything smaller than a sentence.
- Include references won't be rendered in the GitHub rendered view of your article, because they rely on Markdig extensions. They'll be rendered only after publication.
- Ensure that all the text in an include file is written in complete sentences or phrases that do not depend on preceding text or following text in the article that references the include file. Ignoring this guidance creates an untranslatable string in the article that breaks the localized experience.
- Don't embed include references within other include files. They are not supported.
- Place media files in a media folder that's specific to the include subdirectory--for instance, the `<repo>`/includes/media folder. The media directory should not contain any images in its root. If the include file does not have images, a corresponding media directory is not required.
- As with regular articles, don't share media between include files. Use a separate file with a unique name for each include file and article. Store the media file in the media folder that's associated with the include file.
- Don't use an include file as the only content of an article. Include files are meant to be supplemental to the content in the rest of the article.

Example:

```
[!INCLUDE[sample include file](../includes/sampleinclude.md)]
```

Selectors

Use selectors in technical articles when you author multiple flavors of the same article, to address differences in implementation across technologies or platforms.

This is typically most applicable to our mobile platform content for developers. There are currently two different types of selectors in Markdig, a single selector and a multi-selector.

Because the same selector Markdown goes in each article in the selection, we recommend placing the selector for your article in an include file. Then you can reference that include file in all your articles that use the same selector.

The following shows an example selector:

```
> [!div class="op_single_selector"]
- [macOS](../docs/core/tutorials/using-on-macos.md)
- [Windows](../docs/core/tutorials/with-visual-studio.md)
```

You can see an example of selectors in action at the Azure docs.

Code include references

Markdig supports advanced inclusion of code in an article, via its code snippet extension. It provides advanced rendering that builds on GFM features such as programming language selection and syntax coloring, plus nice features such as:

- Inclusion of centralized code samples/snippets from an external repository.
- Tabbed UI to show multiple versions of code samples in different languages.

Gotchas and troubleshooting

Alt text

Alt text that contains underscores won't be rendered properly. For example, instead of using this:

```
![ADextension_2FA_Configure_Step4](./media/bogusfilename
/ADextension_2FA_Configure_Step4.PNG)
```

Escape the underscores like this:

```
![ADextension\_2FA\_Configure\_Step4](./media/bogusfilename
/ADextension_2FA_Configure_Step4.PNG)
```

Apostrophes and quotation marks

If you copy from Word into a Markdown editor, the text might contain "smart" (curly) apostrophes or quotation marks. These need to be encoded or changed to basic apostrophes or quotation marks. Otherwise, you end up with things like this when the file is published: Itâ€™s

Here are the encodings for the "smart" versions of these punctuation marks:

- Left (opening) quotation mark: `&#8220;`
- Right (closing) quotation mark: `&#8221;`
- Right (closing) single quotation mark or apostrophe: `&#8217;`
- Left (opening) single quotation mark (rarely used): `&#8216;`

Angle brackets

It is common to use angle brackets to denote a placeholder. When you do this in text (not code), you must encode the angle brackets. Otherwise, Markdown thinks that they're intended to be an HTML tag.

For example, encode `<script name>` as `&lt;script name&gt;`

Markdown flavor

The docs.microsoft.com site backend supports CommonMark compliant Markdown parsed through the Markdig parsing engine. This markdown flavor is mostly compatible with GitHub Flavored Markdown (GFM), as most docs are stored in GitHub and can be edited there. Additional functionality is added through Markdown extensions.

See also:

Markdown resources

- Introduction to Markdown
- Docs Markdown cheat sheet
- GitHub's Markdown Basics

- The Markdown Guide

Is this page helpful?

Yes No
