User Guide

Markdown++

Published date: 04/19/2023

Table of Contents

Markdown++ Source Documents

Introduction

Markdown is a text-based authoring format created by John Gruber. It's a simple, light-weight and robust language that puts emphasis on efficiency and readability.

Quick Links

Getting Started with Markdown

Learning Markdown

Learning Markdown++

Markdown Cheat Sheet

These sections will go into detail on authoring in Markdown and Markdown++, how-tos for syntax features, as well as how to prepare documents for publishing in ePublisher.

Getting Started with Markdown

Markdown is a text-based authoring format. Any text editor can be used to create Markdown documents. Here are some popular ones to try out:

- Notepad++. Simple, lightweight notepad app with syntax highlighting based on file extension.
- Visual Studio Code. Code-centric text editor. Has community-made extensions, including many for Markdown.
- Obsidian. A Markdown-specific note taking app. Has themes, a writing mode, and a preview mode.
- Typora. A Markdown editor with many authoring experience features.

After choosing a text editor, Markdown and Markdown++ documents can be created using the `.md` extension.

Learning Markdown

This section will detail the features of Markdown, how to write them, and how to use them in ePublisher. For quick reference material, see the Markdown Cheat Sheet.

Any text can be Markdown. Common prose parses into Markdown with no issue. Simple text documents can be formatted into Markdown documents quickly and easily because of this.

Quick Links

Paragraphs

The basic organization of block-level text, the paragraph is the building block of a Markdown document.

Syntax

A Paragraph is created by writing any text content on a line. It is the default block-level element, meaning all content is considered a Paragraph if the content does not have any recognizeable block-level syntax.

Basics

Any amount of text will do to create a Paragraph. Start the line with non-space characters to avoid indentation-related parsing issues.

```
Lorem ipsum dolor sit amet, consectetur adipiscing
 elit, sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua.
```

Separate with Empty Lines

Keep an empty line between Paragraphs that should be separated. This is a general good rule of thumb for all Markdown content.

```
Lorem ipsum dolor sit amet, consectetur adipiscing
 elit, sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua.


Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo
 consequat.
```

Multi-Line Paragraphs

Multiple lines not separated by an empty line will be treated as parts of the same Paragraph. The lines will be consolidated and separated by space in the output.

```
Lorem ipsum dolor sit amet, consectetur adipiscing
 elit, sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo
 consequat.
```

Preserve Line Breaks

Ending a line with a space character at the end will preserve the line break within the Paragraph. Useful for poetry, or other types of content where line structure is important.

```
Nature's first green is gold,

Her hardest hue to hold.

Her early leaf's a flower;

But only so an hour.
```

Markdown++

A custom Paragraph Style can be given to a Paragraph using a Markdown++ style tag on the line directly above the Paragraph.

```
<!--style:CustomParagraph-->

Lorem ipsum dolor sit amet, consectetur adipiscing
 elit, sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua.
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Paragraph**

Style Name: **Paragraph**

Property	**Value**
font family	Arial
font size	12pt
line height	1.2em
padding top	0pt
padding bottom	6pt

If a custom style name is assigned to a Paragraph, that style name will still inherit all of the listed default style information.

Titles

Also referred to as a *setext heading*, Titles are useful to communicate the central idea of a document. Titles are most useful as the leading content of a set of text material.

Syntax

Titles are created by writing a single line of content followed by a line containing at least 1 of either `=` or `-` characters. The second line shouldn't contain text other than these two characters.

Basics

The most basic example, a line of content with a following line with a `=` character.

```
My Document Title

=
```

Titles can be written in the same way using `-` characters.

```
My Document Title

-
```

Any Amount of Characters

The amount of `=` or `-` characters that are used is not important. Having a matching amount of characters on both lines can be a nice touch for readability, though.

```
My Document Title

================
```

Markdown++

A custom Paragraph Style can be given to a Title using a Markdown++ style tag on the line directly above the Title.

```
<!--style:CustomTitle-->

My Document Title

=================
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

The style name a Title will get is dependent on the characters used in the second line. **Title 1** is given to Titles that use `=` characters, and **Title 2** is given to Titles that use `-` characters.

Default Style Properties

Style Type: **Paragraph**

Style Name: **Title 1** , **Title 2**

Property	Value
font family	Arial
font size	24pt
font weight	bold
line height	1.2em
padding top	0pt

Property	Value
padding bottom	12pt

Default Style Options

Option	Value
Table of Contents level	1

If a custom style name is assigned to a Title, that style name will still inherit all of the listed default style information.

Headings

Originally named the *ATX heading*, a Heading communicates a central idea for a topic. Headings should contain the main idea for a section, and have useful keywords to make the section easy to find in a search.

Syntax

Headings are created by starting a line of content with the `#` character. The `#` characters and the text content of the Heading need to be separated by a space character. The amount of `#` characters used indicates the level of heading which will be created.

Basics

Create a Heading 1 with a single `#`, a space, and some text.

```
# Heading 1
```

More `#` characters can be added to the Heading to increase the heading level, up to 6.

```
# Heading 1

## Heading 2

### Heading 3

#### Heading 4

##### Heading 5
```

```
###### Heading 6
```

Markdown++

A custom Paragraph Style can be given to a Heading using a Markdown++ style tag on the line directly above the Heading.

```
<!--style:CustomHeading-->

# Heading 1
```

To learn more about Markdown++ tagging, see Learning Markdown++.

Heading Behavior

Heading Alias

Each created Heading gets an alias that can be used to link to it from another place in the publication.

To determine the alias value, ePublisher takes the text of the Heading, lower-cases it, removes all non-alphanumeric characters, and replaces space with `-` characters.

The below Heading will get the alias value `lets-go-to-the-moon`.

```
# Let's Go to the Moon!
```

Any time the text of a Heading is changed, the alias will also change. It's recommended to use a Custom Alias to avoid having to change link paths when Headings change.

ePublisher Style Information

Style Behavior

The style name ePublisher will create for a Heading will is dependent on the number of `#` characters used at the front of

the line. One `#` character creates the style name **Heading 1**, two `#` characters creates **Heading 2**, etc.

Default Style Properties

Style Type: **Paragraph**

Style Name: **Heading 1**, **Heading 2**, **Heading 3**, **Heading 4**, **Heading 5**, **Heading 6**

Heading 1

Property	Value
font family	Arial
font size	21pt
font weight	bold
line height	1.2em
padding top	0pt
padding bottom	12pt

Heading 2

Property	Value
font family	Arial
font size	18pt
font weight	bold
line height	1.2em
padding top	0pt
padding bottom	12pt

Heading 3

Property	Value
font family	Arial
font size	15pt
font weight	bold
line height	1.2em
padding top	0pt
padding bottom	12pt

Heading 4, Heading 5, Heading 6

Property	Value
font family	Arial
font size	12pt
font weight	bold
line height	1.2em
padding top	0pt
padding bottom	12pt

Default Style Options

Heading 1

Option	Value
Table of Contents level	2

Heading 2

Option	Value
Table of Contents level	3

Heading 3

Option	Value
Table of Contents level	4

Heading 4

Option	Value
Table of Contents level	5

Heading 5

Option	Value
Table of Contents level	6

Heading 6

Option	Value
Table of Contents level	none

If a custom style name is assigned to a Heading, that style name will still inherit all of the listed default style information for the matching Heading syntax.

Lists

Lists are a structural feature in Markdown. They're useful for many things, such as itemizing a collection of information, providing steps to a procedure, or numbering sections of information.

Syntax

There are two types of lists that can be created: Ordered Lists and Unordered Lists.

Ordered Lists are created by starting a line with a number or letter, followed by a single `.`, then space (two is recommended), then some text content.

Unordered Lists are created by starting a line with any `-`, `*`, or `+` character, followed by a space, then some text content.

Beyond these differences, both types of Lists have the same behavior when it comes to syntax and authoring.

Basics

Ordered List

Create a simple Ordered List using a number and a `.` character. Each list item is written on it's own line.

```
1.  list item one

2.  list item two

3.  list item three
```

Ordered Lists can also be created using letters and `.`.

```
a.  list item one

b.  list item two
```

```
c.  list item three
```

Roman numerals are fine, too. Remember to keep the vertical spacing consistent.

```
i.   list item one

ii.  list item two

iii. list item three
```

Re-using the same letter or number is OK.

```
1.  list item one

1.  list item two

1.  list item three
```

```
a.  list item one

a.  list item two

a.  list item three
```

Unordered List

Create a simple Unordered List using `-`. Each list item is written on it's own line.

```
- list item one

- list item two

- list item three
```

Unordered Lists can also be created using `*`.

```
* list item one

* list item two
```

```
* list item three
```

The `+` character can be used as well.

```
+ list item one

+ list item two

+ list item three
```

One Empty Line Between List Items

Put a single empty line between list items to give room. More than one empty line will break the list into two.

```
- list item one


- list item two


- list item three
```

Don't Use Unlike Characters

Using non-matching characters on the same list level will break the list in two.

```
- list item one

* list item one

+ list item one
```

```
1.  list item one

a.  list item one
```

Multi-Line Content in List Items

List Item content can span multiple lines. Use a blank line to separate elements. Make sure all lines of content retain the same vertical spacing.

```
1.  ### Cities in the US


    Here is a sample of some cities in the United
 States.


    | Name   | State    |

    |--------|----------|

    | Austin | Texas    |

    | Tulsa  | Oklahoma |


2.  list item two
```

```
- ### Cities in the US


  Here is a sample of some cities in the United
 States.


  | Name   | State    |

  |--------|----------|

  | Austin | Texas    |

  | Tulsa  | Oklahoma |


- list item two
```

Nested List Items

To nest List items, make sure the vertical spacing of the nested List item matches up with the content of the parent List item.

```
1.  list item one

    1.  nested list item one

2.  list item two

3.  list item three
```

```
- list item one

  - nested list item one

- list item two

- list item three
```

Nesting Different Types of Lists

Nesting Lists of different types is acceptable. Use the same spacing rules as usual.

```
1.  list item one

    - nested list item one

    - nested list item two

2.  list item two

3.  list item three
```

```
- list item one
```

```
  1.  nested list item one


  2.  nested list item two


- list item two


- list item three
```

Markdown++

A custom Paragraph Style can be given to a List using a Markdown++ style tag on the line directly above the List.

```
<!--style:CustomOList-->

1.  A customized ordered list, style name
 "CustomOList"

2.  CustomOList item two

3.  CustomOList item three
```

```
<!--style:CustomUList-->

- A customized unordered list, style name
 "CustomUList"

- CustomUList item two

- CustomUList item three
```

Customizing Nested Lists

Nested Lists can be customized as well.

```
1.  A default list, style name "OList"


    <!--style:CustomOList-->
```

```
    a.  A customized list, style name "CustomOList"

    b.  CustomOList item two

2.  OList item two
```

```
- A default list, style name "UList"

  <!--style:CustomUList-->
  - A customized list, style name "CustomUList"

  - CustomUList item two

- UList item two
```

Default Until Customized

Nested Lists are treated as standalone; they will not inherit the outermost stylename if customized.

```
<!--style:CustomOList-->
1.  A customized list, style name "CustomOList"

    a.  A default list, style name "OList"

    b.  OList item two

2.  CustomOList item two
```

```
<!--style:CustomUList-->
- A customized list, style name "CustomUList"

  - A default list, style name "UList"
```

```
  - UList item two


- CustomUList item two
```

Add a style tag to each list individually for consistency with custom styles.

```
<!--style:CustomOList-->

1.  A customized list, style name "CustomOList"


    <!--style:CustomOList-->

    a.  A customized list, style name "CustomOList"


    b.  CustomOList item two


2.  CustomOList item two
```

```
<!--style:CustomUList-->

- A customized list, style name "CustomUList"


  <!--style:CustomUList-->

  - A customized list, style name "CustomUList"


  - CustomUList item two


- CustomUList item two
```

Nested Content in Lists

The tagging convention can be used for other Markdown elements inside List items. The resulting style name will be appended with the style name of the containing List.

```
1.  <!--style:CustomParagraph-->

    A customized paragraph, style name "OList
 CustomParagraph"


2.  list item two
```

```
- <!--style:CustomParagraph-->

  A customized paragraph, style name "UList
 CustomParagraph"


- list item two
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

To allow full styling of Lists, ePublisher creates a number of style names when a list is detected inside a Markdown source document.

List Style

The List Style is the first style that ePublisher adds to the Style Designer when a list is detected in a source document. The default name is **OList** for ordered lists, and **UList** for unordered lists, but could also be a custom name if the style tag syntax is used on the list.

This style applies to the container area surrounding the lists's items. It's style rules can also apply to list items or nested content, if the same rule isn't already applied on a nested style.

Customizing the List Style

By adding a Markdown++ custom style tag, the List Style name can be changed. The example below changes the List Style name to **CustomUList** :

```
<!--style:CustomUList-->

- This is a custom list

- unordered

- named "CustomUList"
```

List Item Style

The list items inside a list also get a style name. To determine the List Item Style's name, ePublisher takes the List Style and adds `Item` to the end, separated by a space. The default name is **OList Item** for ordered list items, and **UList Item** for unordered list items, but could also be a custom name if the style tag syntax is used on the list.

Customizing the List Item Style

By adding a Markdown++ custom style tag, the List Item Style name can be changed. The example below adds the List Item Style **CustomUList Item** , because the List Style name has been set to **CustomUList** :

```
<!--style:CustomUList-->

- This is a custom list

- unordered

- named "CustomUList"
```

List items can't be styled individually. This will break the list into two separate lists. All list items in a given list are styled by the same List Item Style.

Nested Styles

Nested content inside of list items also get a new style name. To determine the Nested Style's name, take the List Style and add

the style name of the nested content to the end, separated by a space.

The example below populates the Style Designer with 3 Paragraph Styles: **UList** (the List Style), **UList Item** (the List Item Style), and **UList Paragraph** when scanned into ePublisher.

```
- This is a simple list

- unordered

- default style names
```

Customizing Nested Styles

By adding a Markdown++ custom style tag, the Nested Style name can be changed. The example below changes the Nested Style name to **UList CustomParagraph** :

```
- <!--style:CustomParagraph-->

  This is a custom paragraph.
```

Custom Style Names can be used on both the list and nested content simultaneously. This example creates the style names **CustomUList** , **CustomUList Item** , and **CustomUList CustomParagraph** :

```
<!--style:CustomUList-->

- <!--style:CustomParagraph-->

  This is a custom paragraph inside a blockquote.
```

Default Style Properties

Style Type: **Paragraph**

Style Name: **OList** , **OList Item** , **UList** , **UList Item**

OList

Property	Value
padding top	0pt
padding right	0pt
padding bottom	0pt
padding left	0pt
margin top	0pt
margin right	0pt
margin bottom	0pt
margin left	0pt
tag	ol

UList

Property	Value
padding top	0pt
padding right	0pt
padding bottom	0pt
padding left	0pt
margin top	0pt
margin right	0pt
margin bottom	0pt
margin left	0pt
tag	ul

OList Item, UList Item

Property	Value
padding top	0pt
padding right	0pt
padding bottom	0pt
padding left	0pt
margin top	0pt
margin right	0pt
margin bottom	0pt
margin left	36pt
tag	li

If a custom style name is assigned to a List, that style name will still inherit all of the listed default style information.

Tables

Tables lay out multiple lines of detailed data in an organized way. In Markdown, Tables are used to display cells of inline content. This often means that table structue is kept simple.

If a Table with complex structure is needed, it can be created as an HTML fragment in a Block HTML element.

Syntax

Markdown Tables consist of 3 things:

- A **header row**, which contains header cell content separated by `|` characters.
- An **alignment row**, that indicates the alignment of the body cells' text. Each cell in this row contains at least 3 `-` characters, and an optional `:` character to indicate alignment. Each cell is separated by a `|` character.
    - Default alignment only uses `-` characters; 3 or more.
    - Left align the column by starting the cell with `:` and filling in the rest with `:` characters; 3 or more.
    - Right align the column by starting the cell with 3 or more `-` characters, ending with a `:` character.
    - Center align the column by starting and ending the cell with `:` characters. Put `-` characters between them; 3 or more.
- 1 or more **body rows**, that contain body cell content separated by `|` characters.

Each row's content should be confined to a single line. The table will not parse properly if rows have multi-line content.

Optionally, all lines in the table can start and end with `|` characters. Be sure to apply them to all lines if they are to be used.

Basics

Two basic Tables; one with wrapping `|` characters, one without.

```
| name | age | city |
|---|---|---|
| Bob | 42 | Dallas |
| Mary | 37 | El Paso |
```

```
 name | age | city
---|---|---
 Bob | 42 | Dallas
 Mary | 37 | El Paso
```

Line up the `|` characters in each row for a nice touch for readibilty.

```
| name | age | city    |
|------|-----|---------|
| Bob  | 42  | Dallas  |
| Mary | 37  | El Paso |
```

```
 name | age | city
------|-----|------
 Bob  | 42  | Dallas
 Mary | 37  | El Paso
```

Left-align the text of cells in a column by starting the alignment cell with `:`. The first column is left-aligned in this example:

```
| name | age | city    |
|:-----|-----|---------|
| Bob  | 42  | Dallas  |
| Mary | 37  | El Paso |
```

```
 name | age | city
:-----|-----|------
 Bob  | 42  | Dallas
 Mary | 37  | El Paso
```

Right-align the text of cells in a column by ending the alignment cell with `:`. The first column is right-aligned in this example:

```
| name | age | city    |
|-----:|-----|---------|
| Bob  | 42  | Dallas  |
| Mary | 37  | El Paso |
```

```
 name | age | city
-----:|-----|------
 Bob  | 42  | Dallas
 Mary | 37  | El Paso
```

Center-align the text of cells in a column by starting and ending the alignment cell with `:`. The first column is center-aligned in this example:

```
| name | age | city    |
|:----:|:---:|:-------:|
| Bob  | 42  | Dallas  |
| Mary | 37  | El Paso |
```

```
 name | age | city
:----:|:---:|:------
 Bob  | 42  | Dallas
 Mary | 37  | El Paso
```

Each column gets it's own alignment. Mix them together as needed.

```
| name | age | city    |
|:-----|:---:|--------:|
| Bob  | 42  | Dallas  |
| Mary | 37  | El Paso |
```

```
 name | age | city
:-----|:---:|--------:
 Bob  | 42  | Dallas
 Mary | 37  | El Paso
```

Markdown In Tables

Inline Markdown elements, like bold, italic, and even inline HTML, can be used with cell text content.

```
| name     | age | city    |
```

```
|----------|-----|---------|

| **Bob**  | 42  | Dallas  |

| **Mary** | 37  | El Paso |
```

```
 name     | age | city

----------|-----|---------

 **Bob**  | 42  | Dallas

 **Mary** | 37  | El Paso
```

Markdown++

A custom Table Style can be given to a Table using a Markdown ++ style tag on the line directly above the Table.

```
<!--style:CustomTable-->

| name | age | city    |

|------|-----|---------|

| Bob  | 42  | Dallas  |

| Mary | 37  | El Paso |
```

```
<!--style:CustomTable-->

 name | age | city

------|-----|------

 Bob  | 42  | Dallas

 Mary | 37  | El Paso
```

Content in Cells

Inline text content can be customized using the inline tag convention.

```
| name                            | age | city     |
|---------------------------------|-----|----------|
| <!--style:CustomText-->*Bob*    | 42  | Dallas   |
| <!--style:CustomText-->*Mary*   | 37  | El Paso  |
```

```
 name                            | age | city
--------------------------------|-----|------
 <!--style:CustomText-->*Bob*    | 42  | Dallas
 <!--style:CustomText-->*Mary*   | 37  | El Paso
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

In order to style a Table and it's cells in detail, a few different styles are needed in ePublisher. A Table gets 3 styles when ePublisher detects one in a document.

The example below populates the Style Designer with 1 Table Style called **Table**, and 2 Paragraph Styles: **Table Cell Head**, and **Table Cell Body** when scanned into ePublisher.

```
> # Heading 1 element inside a blockquote
>
> This is a Paragraph element inside of a blockquote.
>
```

Table Style

The Table Style is the first style that ePublisher adds to the Style Designer when a table is detected in a source document. The default name is **Table**, but could also be a custom name if the style tag syntax is used on the table.

This style applies table-specific style rules to the entire table. It is the only style in Markdown++ that creates an entry in the **Table Styles** area in the Style Designer.

Customizing the Table Style

By adding a Markdown++ custom style tag, the Table Style name can be changed. The examples below change the Table Style name to **CustomTable**:

```
<!--style:CustomTable-->

| name | age | city    |

|------|-----|---------|

| Bob  | 42  | Dallas  |

| Mary | 37  | El Paso |
```

```
<!--style:CustomTable-->

 name | age | city

------|-----|------

 Bob  | 42  | Dallas

 Mary | 37  | El Paso
```

Header & Body Cell Styles

Every cell on a header row gets a Header Cell Style. Each cell on body rows get a Body Cell Style as well. To determine the Header Cell Style's name, ePublisher takes the Table Style and adds `Cell Head` to the end for Header Cell Styles, and `Cell Body` to the end for Body Cell Styles. The default names are **Table Cell Head** and **Table Cell Body**, but these will also be customized if the Table Style has a custom name.

Customizing Header & Body Cell Styles

By adding a Markdown++ custom style tag, the Header & Body Cell Style names can be changed. The example below changes the style names to **CustomTable Cell Head** and **CustomTable Cell Body** because the Table Style has been given the custom style name **CustomTable**:

```
<!--style:CustomTable-->

| name | age | city    |

|------|-----|---------|

| Bob  | 42  | Dallas  |

| Mary | 37  | El Paso |
```

```
<!--style:CustomTable-->

 name | age | city

------|-----|------

 Bob  | 42  | Dallas

 Mary | 37  | El Paso
```

Default Style Properties

Style Type: **Table**, **Paragraph**

Style Name: **Table**, **Table Cell Head**, **Table Cell Body**

Table

Property	**Value**
border top color	#222222
border top style	solid
border top width	1px

Property	Value
border right color	#222222
border right style	solid
border right width	1px
border bottom color	#222222
border bottom style	solid
border bottom width	1px
border left color	#222222
border left style	solid
border left width	1px

Table Cell Head

Property	Value
font family	Arial
font size	11pt
font weight	bold
padding top	6pt
padding right	6pt
padding bottom	6pt
padding left	6pt

Table Cell Body

Property	Value
font family	Arial

Property	Value
font size	11pt
padding top	6pt
padding right	6pt
padding bottom	6pt
padding left	6pt

If a custom style name is assigned to a Table, the style names will still inherit all of the listed default style information.

Blockquotes

Blockquotes are unique block-level elements that can contain other block-level elements. They have a diverse set of usages due to this, such as capturing a sequence of conversation, or being a container for an important presentation of concepts.

Syntax

Blockquotes are created by starting a line with the `>` character. A space between text content and the `>` character is optional, but recommended. Any Markdown or Markdown++ convention is acceptable as text content inside of Blockquotes.

Basics

A basic Blockquote containing a single Paragraph

```
> A paragraph inside a blockquote.
```

Markdown In Blockquotes

Other Markdown elements, like headings and lists, can be used inside Blockquotes. Use the same spacing and indentation rules as usual when inside Blockquotes.

```
> ### How to Publish Content with ePublisher

>

> Here's some steps to publish your content with
 ePublisher.

>

> 1. Open ePublisher

> 2. Add source documents

> 3. Select Format
```

```
> 4. Click **Generate All**
```

Nested Blockquotes

Other Blockquotes can also be nested inside of Blockquotes, and so on.

```
> First level blockquote

>

> > Second level nested blockquote.

> >

> > > Third level nested blockquote.

> > >
```

Markdown++

A custom Paragraph Style can be given to a Blockquote using a Markdown++ style tag on the line directly above the Blockquote.

```
<!--style:CustomBlockquote-->

> A customized blockquote, style name
 "CustomBlockquote"
```

Customizing Nested Blockquotes

Nested Blockquotes can be customized as well.

```
> A default blockquote, style name "Blockquote"

>

> <!--style:CustomBlockquote-->

> > A customized blockquote, style name
 "CustomBlockquote"
```

```
> >
```

Default Until Customized

Nested Blockquotes are treated as standalone; they will not inherit the outermost stylename if customized.

```
<!--style:CustomBlockquote-->

> A customized blockquote, style name
 "CustomBlockquote"

>

> > A default blockquote, style name "Blockquote"

> >
```

Add a style tag to each blockquote individually for consistency with custom styles.

```
<!--style:CustomBlockquote-->

> A customized blockquote, style name
 "CustomBlockquote"

>

> <!--style:CustomBlockquote-->

> > A customized blockquote, style name
 "CustomBlockquote"

> >
```

Markdown in Blockquotes

The tagging convention can be used for other Markdown elements inside Blockquotes. These style names will inherit the Blockquote's style name as a prefix. See Nested Styles for more info.

```
> <!--style:CustomParagraph-->
```

```
> A customized paragraph, style name "Blockquote
 CustomParagraph"

>

> <!--style:CustomList-->

> - an unordered list

> - customized

> - style name "Blockquote CustomList"

>
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

Blockquotes are considered containers; they *contain* other block-level elements, like Paragraphs, Lists, and Tables. Because of this, ePublisher creates a number of different styles when it detects blockquotes in source documents.

Blockquote Style

The Blockquote Style is the first style that ePublisher adds to the Style Designer when a blockquote is detected in a source document. The default name is **Blockquote**, but could also be a custom name if the style tag syntax is used on the blockquote.

This style applies to the container area surrounding the blockquote's content. It's style rules can also apply to nested content, if the same rule isn't already applied on the nested style.

Customizing the Blockquote Style

By adding a Markdown++ custom style tag, the Blockquote Style name can be changed. The example below changes the Blockquote Style name to **CustomBlockquote**:

```
<!--style:CustomBlockquote-->

> This is a custom named blockquote.

>
```

Nested Styles

Nested content inside of Blockquotes also get a new style name. To determine the Nested Style's name, take the Blockquote Style and add the style name of the nested content to the end, separated by a space.

The example below populates the Style Designer with 3 Paragraph Styles: **Blockquote** (the Blockquote Style), **Blockquote Heading 1** , and **Blockquote Paragraph** when scanned into ePublisher.

```
> # Heading 1 element inside a blockquote

>

> This is a Paragraph element inside of a blockquote.

>
```

Customizing Nested Styles

By adding a Markdown++ custom style tag, the Nested Style name can be changed. The example below changes the Nested Style name to **Blockquote CustomParagraph** :

```
> <!--style:CustomParagraph-->

> This is a custom paragraph.

>
```

Custom Style Names can be used on both the blockquote and nested content simultaneously. This example creates the style names **CustomBQ** , and **CustomBQ CustomParagraph** :

```
<!--style:CustomBQ-->
```

```
> <!--style:CustomParagraph-->

> This is a custom paragraph inside a blockquote.

>
```

Default Style Properties

Style Type: **Paragraph**

Style Name: **Blockquote**

Property	**Value**
background color	#efefef
border left style	solid
border left color	#DFE2E5
border left width	3pt
padding top	12pt
padding right	12pt
padding bottom	12pt
padding left	12pt

If a custom style name is assigned to a Blockquote, that style name will still inherit all of the listed default style information.

Code Fences

A Code Fence preserves all text content it encapsulates and presents it exactly how it was written. Code Fences are useful for presenting information that needs to be written explicitly, like examples of code. They can provide users with content that can be copied and used for their own purposes, too.

Syntax

Code Fences are created in three steps:

1. Start with a line containing ```` ``` ```` or `~~~`.
2. A following line or lines of text content.
3. End with a line containing ```` ``` ```` or `~~~`, matching the starting line.

Basics

A simple example using ```` ``` ```` tags. Any amount of text can be written between the two tags, as long as the tags match and are written correctly.

````
```

function addTwoNumbers(num1, num2) {

  return num1 + num2;

}

```
````

Code Fences can be created using `~~~`, too.

```
~~~
```

```
function addTwoNumbers(num1, num2) {

  return num1 + num2;

}

~~~
```

No Parsing in Code Fences

Markdown written inside of Code Fences will render as plain text.

```
```

Heading 1 in Plain Text

```
```

HTML will also render as plain text when written inside Code Fences.

```
```

HTML in plain text

```
```

Markdown++

A custom Paragraph Style can be given to a Code Fence using a Markdown++ style tag on the line directly above the Code Fence.

```
<!--style:CustomCodeFence-->

```

function addTwoNumbers(num1, num2) {

  return num1 + num2;

}
```
```

```
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Paragraph**

Default Style Name: **Code Fence**

Property	Value
background color	#efefef
font family	Consolas
font size	11pt
margin top	6pt
margin bottom	6pt
padding top	12pt
padding right	12pt
padding bottom	12pt
padding left	12pt
overflow	auto
white space	pre

If a custom style name is assigned to a Code Fence, that style name will still inherit all of the listed default style information.

Code Blocks

A Code Block preserves all text content it encapsulates and presents it exactly how it was written. Code Blocks are useful for presenting information that needs to be written explicitly, like examples of code. They can provide users with content that can be copied and used for their own purposes, too.

Syntax

Code Blocks are created by adding 4 spaces before text content. A Code Block can consist of one or more lines created in this manner.

Basics

Starting a line with 4 spaces will create a basic Code Block.

```
    var firstName, lastName;
```

Multi-Line Code Blocks

Multiple lines can be used; start all lines with at least 4 spaces.

```
    var firstName, lastName;



    firstName = "John";

    lastName = "Doe";
```

Space is Preserved.

Any spaces after the first 4 will be used as indentation for the content of the Code Block.

```
    function addTwoNumbers(num1, num2) {
```

```
  return num1 + num2;

}
```

No Parsing in Code Blocks

Markdown written inside of Code Blocks will render as plain text.

```
# Heading 1 in Plain Text
```

HTML will also render as plain text when written inside Code Blocks.

```
<p>HTML in plain text</p>
```

Markdown++

A custom Paragraph Style can be given to a Code Block using a Markdown++ style tag on the line directly above the Code Block.

```
<!--style:CustomCodeBlock-->

function addTwoNumbers(num1, num2) {

  return num1 + num2;

}
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Paragraph**

Style Name: **Code Block**

Property	Value
background color	#efefef
font family	Consolas
font size	11pt
margin top	6pt
margin bottom	6pt
padding top	12pt
padding right	12pt
padding bottom	12pt
padding left	12pt
overflow	auto
white space	pre

If a custom style name is assigned to a Code Block, that style name will still inherit all of the listed default style information.

Horizontal Rules

A Horizontal Rule provides a visual separation between sections of content. They're useful to separate unrelated ideas on a single page.

Syntax

A Horizontal Rule is created by using at least 3 `-`, `_`, or `*` characters. These should be the only characters on the line, but any combination of them is acceptable.

Basics

A simple Horizontal Rule using `-` characters.

```
---
```

An example using `*` characters.

```
***
```

And one with `_` characters.

```
___
```

3 or More Characters

More than 3 characters can be used, if desired.

```
----------
```

Spaces OK

Spaces are acceptable between the characters.

```
- - - - - - -
```

Mixed Characters

Combinations of the 3 characters is fine to use as well.

```
-*_*-*_*-*_*-
```

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

```
* - * - * - *
```

Markdown++

A custom Paragraph Style can be given to a Horizontal Rule using a Markdown++ style tag on the line directly above the Horizontal Rule.

```
<!--style:CustomHR-->

---
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Paragraph**

Style Name: **Horizontal Rule**

Property	Value
border top color	#222222

Property	Value
border top style	inset
border top width	1px
border right color	#222222
border right style	inset
border right width	1px
border bottom color	#222222
border bottom style	inset
border bottom width	1px
border left color	#222222
border left style	inset
border left width	1px
display	block
margin top	6pt
margin bottom	6pt
tag	hr

If a custom style name is assigned to a Horizontal Rule, that style name will still inherit all of the listed default style information.

Block HTML

The common markup language for web technology, HTML, can be used in Markdown documents on the block level. Refer to W3Schools' HTML Tutorial to learn more about how to write and use HTML.

Syntax

Block HTML is created by writing a valid HTML fragment on a line or set of lines. HTML syntax must be the first thing on the line to be considered Block HTML.

Basics

Simple Block HTML using a `p` element.

```
<p>A simple paragraph element.</p>
```

Multi-Line HTML

HTML can span multiple lines. Keep it compact. An empty line will break the fragment in two, so it is best used to separate the fragment from other content.

```
<table>

  <tr>

    <th>Name</th>

    <th>Age</th>

    <th>Country</th>

  </tr>

  <tr>

    <td>John Doe</td>
```

```
    <td>35</td>
    <td>USA</td>
  </tr>
  <tr>
    <td>Jane Doe</td>
    <td>32</td>
    <td>USA</td>
  </tr>
</table>
```

No Markdown in Block HTML

Markdown syntax can't be used inside of Block HTML. The entire HTML fragment is passed straight to the output as-is.

```
<p>No **Markdown** here.</p>
```

Markdown++

A custom Paragraph Style can be given to Block HTML using a Markdown++ style tag on the line directly above the Block HTML.

```
<!--style:CustomHTML-->

<p>HTML block given the style name "CustomHTML"</p>
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

All HTML fragments are wrapped in a container element, which is given a style name. The default name is **HTML**, but can also be a custom name if the style tag is used directly above an HTML fragment.

HTML is unavailable for publishing in PDF or PDF XSL-FO output due to incompatibility with those technologies. ePublisher will remove any HTML content it detects before generating PDF output.

Default Style Properties

Style Type: **Paragraph**

Style Name: **HTML**

Property	**Value**
display	block
overflow	auto

If a custom style name is assigned to a Block HTML, that style name will still inherit all of the listed default style information.

Bold, Italic, Strikethrough, Code

Inline text can be styled to put emphasis or formatting on certain phrases. Markdown offers wrappers for Bold, Italic, Strikethrough, and Code.

Syntax

Bold text is created by wrapping a set of text with a pair of either `**` or `__` characters.

Italic text is created by wrapping text with a pair of either `*` or `_`.

Strikethrough text is created by wrapping text between a pair of `~~` characters.

Code spans are created by wrapping text between a pair of `` ` `` characters.

Basics

Two simple examples for Bold text. Notice either `*` or `_` can be used, but there must be two on each side of the wrap. The start and end characters must also match.

```
Here's **bold** and here's also __bold__.
```

Italic text is written similarly, using one `*` or `_` instead of two.

```
Here's *italic* and here's also _italic_.
```

Same rules apply to Strikethrough text, using `~~`.

```
Using ~~strikethrough~~ text.
```

Code spans follow the same rules, too.

```
Defining a `technical term`.
```

Mixing Styles of Text

Combinations of these can be used together. Make sure the innermost pair of tags is closed before closing an outer pair. Using unlike characters for different pairs helps with readability. (Using `*` for bold, `_` for italic, etc.)

```
We can write **bold and _italic_**.
```

Spanning Multiple Lines

Inline text decorators can span multiple lines, as long as there are no empty lines between the start and end tags.

```
Writing a sentence that **has

bold text** across lines.
```

Markdown++

A custom Character Style can be given to Inline Text using a Markdown++ style tag directly before the start tag of the Inline Text.

```
Styling <!--style:CustomBold-->**inline text. Style
 name "CustomBold"**.
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Character**

Style Name: **Bold** , **Italic** , **Strikethrough** , **Code**

Bold

Property	Value
font weight	bold

Italic

Property	Value
font style	italic

Strikethrough

Property	Value
text decoration	line-through

Code

Property	Value
background color	#efefef
font family	Consolas
white space	pre

If a custom style name is assigned to Inline HTML, that style name will still inherit all of the listed default style information.

Links

Links are an inline Markdown convention used to connect users to other locations and resources in a set of information.

Syntax

Link syntax looks interesting, but is simple enough once written a few times. Write the link's displayed text in between `[` and `]` characters, and directly next to it write the link's URL between `(` and `)` characters. Optionally, a title can be given to the Link, written next to the link URL, separated by a space and wrapped in `"` characters.

Basics

A basic Link example.

```
[Link Text](path/to/my_doc.md)
```

Titles are optional. Keep the URL and title separate with a space. Wrap the title in `"` characters.

```
[Link Text](path/to/my_doc.md "Link Title")
```

Links can be the only thing on a line or mixed in anywhere inline text can go.

```
To see more, follow the [Link](path/to/my_doc.md).
```

Relative paths, absolute paths, web links, and Aliases are all valid path values.

```
[Link Text](../my_doc.md)


[Link Text](D:/Markdown/Docs/my_doc.md)
```

```
[Link Text](https://www.webworks.com)


[Link Text](#my-doc)
```

Using Link References

Links can make use of Link References to simplify URL management for documents with many different link paths.

```
[Link Text][0]


[0]: my_image.png
```

Titles are also available and written the same way using Link References.

```
[Link Text][0]


[0]: my_image.png "Link Title"
```

Markdown++

A custom Character Style can be given to an Image using a Markdown++ style tag immediately before the Link syntax.

```
<!--style:CustomLink-->[Link Text](path/to/my_doc.md)
```

To learn more about Markdown++ tagging, see Learning Markdown++.

Link Behavior

Links in ePublisher have a variety of ways to connect to other resources in a publication. What they connect to depends on what is used as a path value. All path values inside links also apply to path values in Link References.

Web Links

Write a fully qualified web URL in the path area to link to an external resource. Make sure to start the URL with `http://` or `https://`.

```
[WebWorks Website](https://www.webworks.com)
```

Link to Other Documents

Write the file path for the intended file in the path area to link to another document. Relative paths need to resolve from the file the link is written in. Absolute paths can also be used.

```
[link text](path/to/my_doc.md)
```

```
[link text](C:/Users/me/path/to/my_doc.md)
```

Link to Topics in Other Documents

To link to a specific section in a document, write the file path followed immediately with the alias for the topic. This can be either a Heading Alias or a Custom Alias. Relative paths need to resolve from the file the link is written in. Absolute paths can also be used.

The examples link to the alias `#my-alias` in `my_doc.md`.

```
[link text](path/to/my_doc.md#my-alias)
```

```
[link text](C:/Users/me/path/to/my_doc.md#my-alias)
```

Link to Topics in Same Document

Use an alias by itself to link to a topic in the current document. This can be either a Heading Alias or a Custom Alias.

The example links to the alias `#my-alias` in the current document.

```
[link text](#my-alias)
```

ePublisher Style Information

Default Style Properties

Style Type: **Character**

Style Name: **Link**

Property	Value
text decoration	underline
color	#0078d7

If a custom style name is assigned to a Link, that style name will still inherit all of the listed default style information.

Images

Images are an inline Markdown convention used to display graphics in a document.

Syntax

The image syntax is identical to the Link syntax, with the addition of a `!` character at the beginning. Alt text is written between `![` and `]` characters. Inside of `(` and `)`, the URL path to the image should be written, then, optionally, a title wrapped in `"` characters.

Basics

A basic Image example.

```
![alt text](path/to/my_image.png)
```

Titles are optional. Keep the URL and title separate with a space.

```
![alt text](path/to/my_image.png "Image Title")
```

Images can be the only thing on a line or mixed in anywhere inline text can go.

```
Images can go anywhere text can: ![alt text](path/to/
my_image.png)
```

Relative paths and absolute paths can both be used.

```
![alt text](../my_image.png)


![alt text](D:/Images/my_image.png)
```

Using Link References

Images can also make use of [Link References][md-link-reference] in the same way Links do.

```
![alt text][0]


[0]: my_image.png
```

Titles are also available and written the same way using Link References.

```
![alt text][0]


[0]: my_image.png "Image Title"
```

Markdown++

A custom Graphic Style can be given to an Image using a Markdown++ style tag immediately before the Image syntax.

```
<!--style:CustomImage-->![alt text](path/to/
my_image.png)
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Default Style Properties

Style Type: **Graphic**

Style Name: **Image**

If a custom style name is assigned to an Image, that style name will still inherit all of the listed default style information.

Link References

Link References accompany Links and Images, and are used to keep path values in a separated location from text content. They're useful for readability because they simplify links and images in inline text, and can be written in a standalone location for editing en masse.

This section requires familiarity with Links and Images to teach referencing concepts.

Syntax

A Link Reference must be the only thing on a line. The **link key** is written between `[` and `]:`. The URL path is written next, separated by a space. Optionally, a title can be written after the URL, separated by a space.

Once a Link Reference has been written, the **link key** from it can be used with a Link or Image. Replace the Link/Image's parenthesis `()` section with the link key between `[` and `]`.

Basics

A Link Reference example used with an accompanying Link. The Link is written first and makes use of the link key, in this example `0`.

```
[Link Text][0]


[0]: path/to/my_doc.md
```

Titles are optional. Keep the URL and title separate with a space.

```
[Link Text][0]


[0]: path/to/my_doc.md "Link Title"
```

A Link Reference example with an accompanying Image.

```
![alt text][0]


[0]: path/to/my_image.png
```

Make sure to keep the Reference on it's own line. The Link or Image can be used anywhere text is allowed, though.

```
For more info, check the [Link][0].


[0]: my_doc.md
```

Use Unique Values for Link Keys

Any text will work for the link key, but something unique that can be searched for will help in the authoring process. Link keys must be one-of-a-kind as well. In the case of overlapping link keys, the last link key written will be the accepted one.

```
[wwdoc_0001]: my_doc.md

[wwdoc_0002]: doc2.md

[wwdoc_0003]: doc3.md


[wwimg_0001]: img1.png

[wwimg_0002]: img2.png

[wwimg_0003]: img3.png
```

Inline HTML

The common markup language for web technology, HTML, can be used in Markdown documents mixed with text and other inline elements. Refer to W3Schools' HTML Tutorial to learn more about how to write and use HTML.

Syntax

Inline HTML is created by writing a valid HTML fragment in an area where other inline content exists.

Basics

Simple Inline HTML using a `strong` element.

```
Write words with <strong>bold</strong> emphasis.
```

Markdown and Inline HTML

Markdown syntax can be mixed with Inline HTML.

```
We can <span>write **bold** text</span>.
```

Markdown++

A custom Character Style can be given to Inline HTML using a Markdown++ style tag directly before the Inline HTML.

```
Styling <!--style:CustomHTML--><span>inline HTML.
 Style name "CustomHTML".</span>
```

To learn more about Markdown++ tagging, see Learning Markdown++.

ePublisher Style Information

Style Behavior

All HTML fragments are wrapped in a container element, which is given a style name. The default name is **HTML**, but can also be a custom name if the style tag is used directly before an HTML fragment.

HTML is unavailable for publishing in PDF or PDF XSL-FO output due to incompatibility with those technologies. ePublisher will remove any HTML content it detects before generating PDF output.

Default Style Properties

Style Type: **Character**

Style Name: **HTML**

If a custom style name is assigned to Inline HTML, that style name will still inherit all of the listed default style information.

Learning Markdown++

This section will detail the features of Markdown++, how to write them, and how to use them in ePublisher. For quick reference material, see the Markdown Cheat Sheet.

Markdown++ is a superset of Markdown, meaning that any convention available in Markdown also applies to Markdown++. Learn about Markdown before reading this section.

Quick Links

Markdown++ Basics

Markdown++ is a superset of Markdown. Because of this, *all Markdown files are also Markdown++ files*. Any tools used for Markdown also work well for Markdown++.

Filling out Markdown with a full designing & publishing experience, while also maintaining readability, is a major design goal of Markdown++. Another goal is to preserve the integrity of rendering and previews across the many Markdown tools out there.

Markdown++ uses the HTML Comment tag with a set of commands inside them for most of it's features. Using these enables Markdown++ syntax to be transparent when documents are rendered or previewed, and aids in quick learning by using a well-established pattern.

Learning the HTML Comment tag opens the door to learning most of the features Markdown++ offers.

Syntax

Write an HTML Comment by starting with `<!--` and ending with `-->`. Any text can be written between these two patterns. Keeping the entire comment on a single line is required by Markdown++.

Any unrecognized text inside of a comment gets treated as a regular HTML comment and carries through to the output.

Basics

A simple comment tag with a Custom Style Name command. Keep the tag on a single line.

```
<!--style:CustomStyle-->
```

Apply commands to block-level elements by adding the tag to the line directly above the block element. The style **CustomParagraph** is added to a paragraph below.

```
<!-- style:CustomParagraph -->

A customized paragraph, named "CustomParagraph".
```

Apply commands to inline elements by adding the tag directly before the inline syntax. Don't put space between the comment tag and the inline syntax. The style **CustomBold** is added to bold text below.

```
Customizing some <!--style:CustomBold-->**bold text**.
```

Whitespace OK

In general, it is safe to include any amount of whitespace *between the comment tags*. Use it as necessary for readability.

```
<!-- style: CustomStyle -->
```

Multiple Commands

Any number of commands can be put inside the comment. Separate the commands with a `;` character. This example applies two commands to a Heading 1: a Custom Style Name, and a Custom Alias.

```
<!-- style:CustomHeading1 ; #custom-heading1 -->

# Heading 1
```

Start & End Tags

Some features, like Conditional Text, require start & end tags. The example wraps condition tags around content meant only for printed publications.

```
<!--condition:print_only-->

## Print Only


This ccontent is meant for print only.
```

```
<!--/condition-->
```

Custom Styles

The Custom Style command overrides the default Style Name of a Markdown element with a user-defined Style Name. Using this feature enables a virtually limitless amount of styles for designing & publishing in ePublisher.

Syntax

The Custom Style command is created by writing `style:` followed by the name of the intended style.

Basics

A basic Custom Style command applied to a Paragraph.

```
<!--style:CustomParagraph-->

This paragraph has it's style name customized to
 "CustomParagraph".
```

Put the tag on the line above any block-level element to customize it. Make sure there are no empty lines between the tag and the block element. The tag needs to be the only thing on it's line.

```
<!--style:CustomHeading1-->

# This Heading has been renamed to "CustomHeading1".
```

For Custom Styles on inline text content, put the tag directly before the starting syntax. No space should be put between the tag and the inline syntax. A string of bold text is customized with the Style Namme **CustomBold** below.

```
This paragraph has customized <!--style:CustomBold--
>**bold text**.
```

Mix with Other Commands

Custom Styles can be in the same comment tag with other commands. Separate them with a `;` character. A Custom Style and Custom Alias are written in the same tag below.

```
<!-- style:CustomStyle ; #custom-alias -->
```

Custom Style Command Behavior

Through the Custom Style command, it is possible to get name entries for almost any type of style into ePublisher's Style Designer. How to do so varies based on style type.

Custom Paragraph Style

Add the style tag to the line directly above a block-level element to give it a custom Paragraph Style. This applies to any block-level element, except for Tables.

```
<!--style:CustomParagraph-->

This is a custom paragraph called "CustomParagraph".
```

```
<!--style:CustomHeading1-->

# This is a custom paragraph called "CustomParagraph".
```

```
<!--style:UList-->

- custom list

- unordered

- called "CustomUList"
```

```
<!--styleCustomBlockquote-->
```

```
> This is a custom blockquote

>
```

```
<!--style:CustomHTML-->

<div>

  <p>This is customized HTML. Named "CustomHTML".</p>

</div>
```

Custom Character Style

Add a style tag directly before inline syntax to create a custom Character Style. Remember, no space between the tag and the inline syntax. This applies to any inline syntax, except for Images.

```
This is customized <!--style:CustomBold-->**bold
 text**.
```

```
This is customized <!--style:CustomItalic-->*italic
 text*.
```

```
This is a customized <!--style:CustomLink-->[link]
(my_doc.md).
```

```
This is a customized <!--style:CustomHTML--
><span>Inline HTML</span>.
```

Custom Graphic Style

Add a style tag directly before image syntax to create a custom Graphic Style. Remember, no space between the tag and the image syntax.

```
<!--style:CustomImage-->![alt text](my_image.png)
```

```
<!--style:CustomImage-->![alt text][link_key]
```

Custom Table Style

Like custom Paragraph Styles, add the tag to the line directly above Table syntax to give it a custom Table Style.

```
<!--style:CustomTable-->

| name | age | city    |

|------|-----|---------|

| Bob  | 42  | Dallas  |

| Mary | 37  | El Paso |
```

```
<!--style:CustomTable-->

 name | age | city

------|-----|------

 Bob  | 42  | Dallas

 Mary | 37  | El Paso
```

Custom Page Style

Custom Page Styles need to be created through the Custom Markers command. Refer to the linked section for details.

```
<!--markers:{"PageStyle": "CustomPage"}-->

# Topic Heading
```

Custom Marker Style

Custom Marker Styles need to be created through the Custom Markers command. Refer to the linked section for details.

```
<!--markers:{"Keywords": "topic, heading, markers"}-->

# Topic Heading
```

Custom Aliases

Use a Custom Alias to give an element a unique pointer for linking to in other places in the publication. The Custom Alias is a powerful tool that can simplify link management and speed up authoring and editing.

Syntax

Create a Custom Alias by starting with a single `#` character, followed any alphanumeric characters, `-`, or `_`. Space characters cannot be used in an alias; the Alias will cut off before the space.

Basics

A basic Custom Alias applied to a Heading 1.

```
<!--#custom-alias-->

# Custom Aliased Heading
```

Inline syntax can be given an Alias as well.

```
This <!--#bold-keyword-->**bold text** has a custom
 alias.
```

Mix with Other Commands

Custom Aliases can be in the same comment tag with other commands. Separate them with a `;` character. A Custom Style and Custom Alias are written in the same tag below.

```
<!-- style:CustomStyle ; #custom-alias -->
```

Custom Alias Behavior

Custom Aliases create an entry in the document's WIF that enable linking to the element it was created with.

Using a Custom Alias

The first step in using a Custom Alias is to create one by adding the Alias tag to the location that will be linked to. Below, the Custom Alias `#my-alias` is created, associated with a Heading 1.

```
<!--#my-alias-->

# A Topic Heading
```

After that, the Alias `#my-alias` can be used in the path value for Links and Link References. Once the link is clicked, it will take readers to the location the Alias was created.

Refer to the linked sections for detailed instructions on using these elements.

Some quick examples for Links:

```
[link text](#my-alias)
```

```
[link text](my_doc.md#my-alias)
```

Some quick examples for Link References:

```
[link text][link_key]


[link_key]: #my-alias
```

```
[link text][link_key]


[link_key]: my_doc.md#my-alias
```

Markers in Markdown++

Markers are pieces of metadata that can be inserted into a document to add different features or change behavior of the publication. Markers are useful for many things, like adding keywords to a topic to improve search relevance, or instructing ePublisher to pass text through to the output without any processing.

Syntax

The Markers command has two main parts. Start the command with `markers:`, and follow it with a JSON Object Literal. The Marker name is written in the object key area, and the Marker value goes into the object value area. Make sure all keys and all values are wrapped in `"`, and separated by `:`. Multiple key/value pairs for Markers can be written, separated with `,`.

Basics

A basic Markers command with a Keywords marker.

```
<!--markers:{"Keywords": "webworks"}-->

# About WebWorks
```

Multiple Markers can be added in one command. Separate the key/value pairs with `,`.

```
<!--markers:{"Keywords": "webworks"}-->

# About WebWorks
```

The Markers command is available for inline-level syntax as well.

```
Add a custom <!--markers:{"Keywords": "inline,
 marker"}-->**marker**.
```

Mix with Other Commands

Markers can be in the same comment tag with other commands. Separate them with a `;` character. A Custom Style and Markers are written in the same tag below.

```
<!-- style:CustomStyle ; markers:{"Keywords":
 "webworks"} -->
```

Markers Behavior

Markers associate a piece of data with an element on the page. To learn more about ePublisher's built-in Markers and what they do, see the Markers reference table.

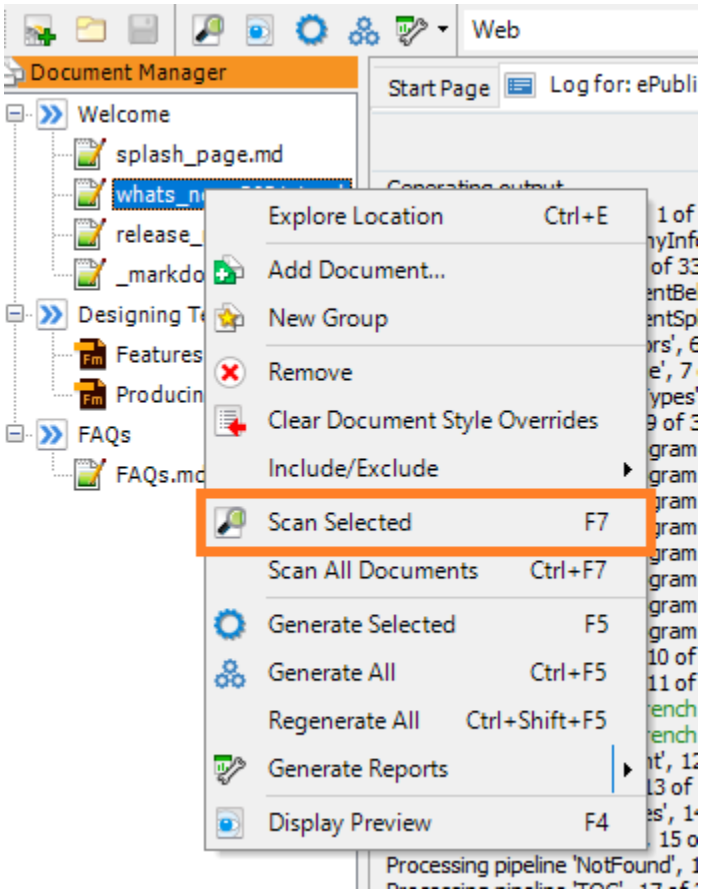
Using a Marker

First thing to do is write the Marker in the intended area of the content. The Marker needs to be tagged to a content element, like a Paragraph or Heading.

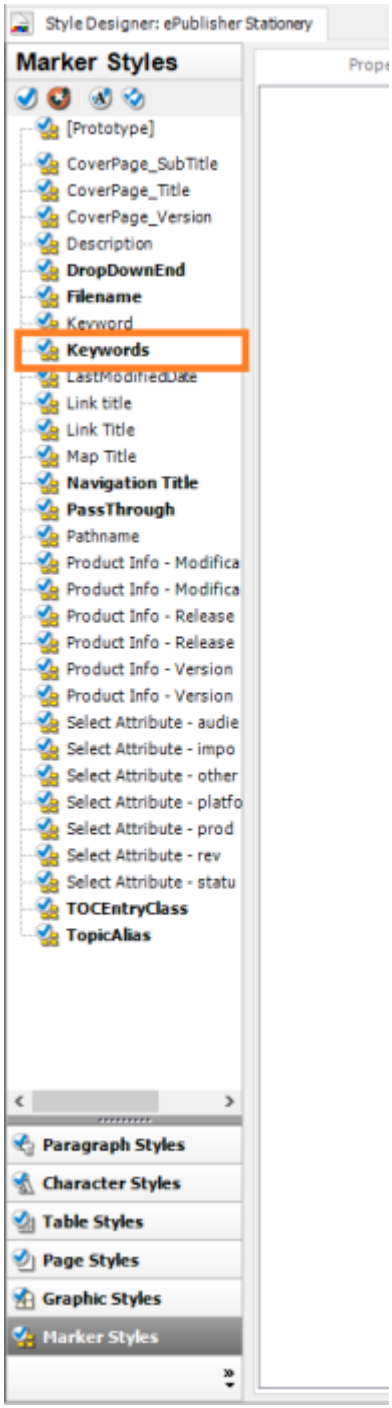
Below, a Keywords Marker is tagged to a Heading 1.

```
<!--markers:{"Keywords": "markers, content, create"}--
>

# Using Markers in Source Content
```

Next, scan the document in ePublisher. This will add the Marker in the Marker Styles area of the Style Designer.



With the Marker added to ePublisher, output can now be generated with new Keywords added. This Marker will improve search relevance in outputs with searching capabilities, like Reverb 2.0.


Style Designer: ePublisher Stationery
Marker Styles
[Prototype]
CoverPage_SubTitle
CoverPage_Title
CoverPage_Version
Description
DropDownEnd
Filename
Keyword
Keywords
LastModifiedDate
Link title
Link Title
Map Title
Navigation Title
PassThrough
Pathname
Product Info - Modifica
Product Info - Modifica
Product Info - Release
Product Info - Release
Product Info - Version
Product Info - Version
Select Attribute - audie
Select Attribute - impo
Select Attribute - other
Select Attribute - platfo
Select Attribute - prod
Select Attribute - rev
Select Attribute - statu
TOCEntryClass
TopicAlias
Paragraph Styles
Character Styles
Table Styles
Page Styles
Graphic Styles
Marker Styles

Conditions

Conditions allow an author to create sections of content that are available in certain contexts, and excluded in others. They can be useful for the Edit/Review process, or to switch out sections meant only for print or the Web.

Syntax

Conditional text is written between two HTML Comment Tags. In the first tag, write one or more conditions between `<!--condition:` and `-->`. The second closing tag is always written as `<!--/condition-->`. Between the tags, any valid Markdown ++ content can be written. Condition names must only use alphanumeric characters, `-`, and `_`. Do not use spaces in Condition names.

See Condition Operators for details on advanced conditional logic syntax.

Basics

A Condition containing a single Paragraph, using the Condition `print_only`.

```
<!--condition:print_only-->

This paragraph is meant only for print.

<!--/condition-->
```

Conditions can contain multiple block-level elements.

```
<!--condition:print_only-->

# The Print Section


This sections is meant only for print.
```

```
It will not be visible if `print_only` is set to
 `Hidden`.

<!--/condition-->
```

Conditions can be used with inline content, too.

```
Go to the Section <!--condition:print_only-->on page
 304<!--/condition--> for more details.
```

Operators

Complex logic can be used with Conditions using a set of Operators. This block is hidden when `production` is set `Visible` using the `!` (logical NOT) operator.

```
<!--condition:!production-->

This paragraph is not meant for production
 publications.

<!--/condition-->
```

Multiple Conditions

Multiple Conditions can be used in a single conditional block Operators. This example only show the block of text if `print_only` AND `production` are set to `Visible`.

```
<!--condition:print_only production-->

This paragraph is meant only for print and production.

<!--/condition-->
```

Conditions Behavior

Conditions are rendered or removed from the document when publishing based on values set in the Conditions Window. Conditions are considered unset, and therefore always render,

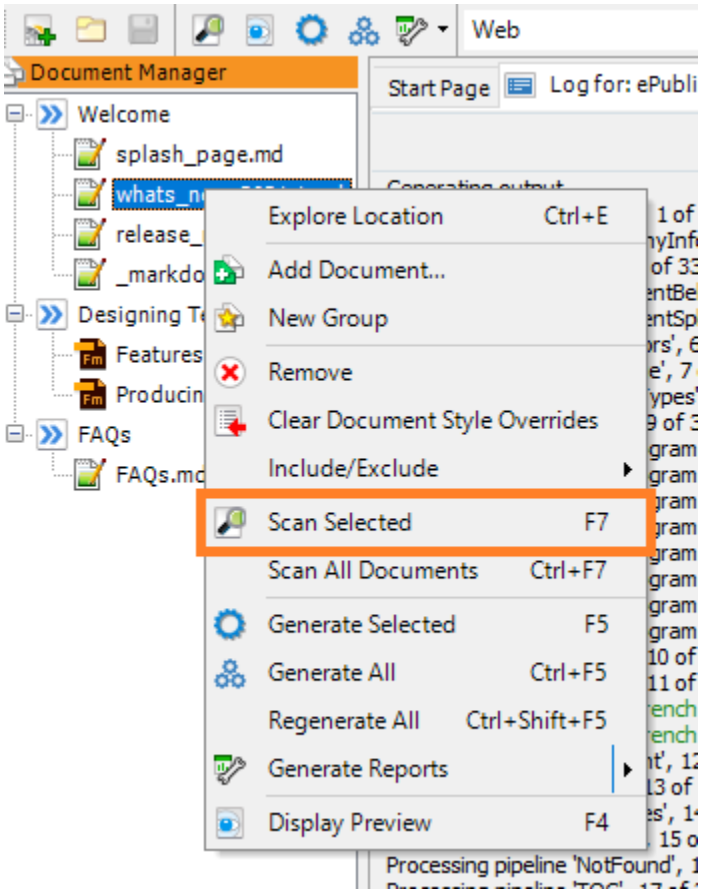
if they haven't been scanned into ePublisher. Conditions are also considered unset if they still have the default value `Use document condition` in the Conditions Window.

Using Conditions

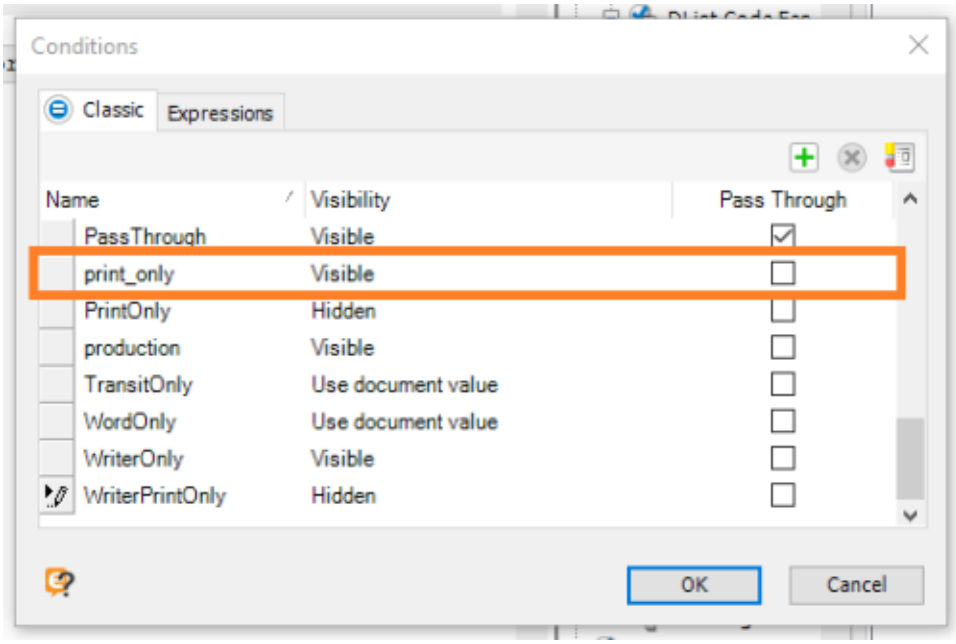
First, create a condition by writing it in a source document. Below, and block of conditional text is created with the Condition `print_only`.

```
<!--condition:print_only-->

# The Print Section


This sections is meant only for print.


It will not be visible if `print_only` is set to
 `Hidden`.

<!--/condition-->
```

Next, scan the document in ePublisher. This will add the Condition `print_only` to the Conditions Window.



Once scanned, the `print_only` Condition's value can be changed to either `Visible` or `Hidden`. The Condition is considered unset, and will always render, if the value is left with the default, `Use document value`.



Condition Operators

Using Operators, an author can create additional logic to determine whether conditional text should be rendered or hidden.

In a conditional statement, the block of text renders if the entire statement evaluates to `true`. The block is hidden if the statement evaluates to `false`.

In this context, `Visible` is considered `true`, and `Hidden` is considered `false`. `Use document value` disables the conditional statement and always renders the block.

Combine Condition names and Operators to create complex statements to determine if the content should be rendered or removed in the publication.

The Space Operator - Logical AND

The space character in a conditional statement is a Logical AND. Meaning, if the statements on both sides of ` ` evaluate to `true`, the statement passes.

The conditional text below is rendered if `print_only` AND `production` are set to `Visible`.

```
<!--condition:print_only production-->

This paragraph is meant only for print and production.

<!--/condition-->
```

The `,` Operator - Logical OR

The `,` character in a conditional statement is a Logical OR. Meaning, if a statement on either side of `,` evaluates to `true`, the statement passes.

The conditional text below is rendered if one of `print_only` OR `production` are set to `Visible`.

```
<!--condition:print_only,production-->

This paragraph is meant for either print or
 production.

<!--/condition-->
```

The `!` Operator - Logical NOT

The `!` character in a conditional statement is a Logical NOT. Adding `!` to the beginning of a Condition reverses it's truthiness. Meaning, a Condition with `!` on the front of it's name evaluates `Visible` to `false` and `Hidden` to `true`.

The conditional text below is removed if `production` is set to `Visible`.

```
<!--condition:!production-->

This paragraph is not meant for production.

<!--/condition-->
```

Conditions and Includes

Conditions can be used as expected in File Includes since they are processed at the same time as Includes. Additionally,

Includes can be used inside of conditions to import entire documents based on certain Conditions.

File Includes

A File Include statement points to another Markdown++ file and imports the file's contents at the location of the statement. This enables multi-file structure in a single document.

Syntax

An Include statement is created by writing a path to a Markdown++ document between `<!--include:` and `-->`. Relative paths and absolute paths are both valid path values. Web paths are not supported. The include statement must be the only thing on a line.

Basics

A basic Include statement. The Include must be written on it's own line to work properly.

```
<!--include:my_file.md-->
```

Relative paths and absolute paths are fine to use.

```
<!--include:my_file.md-->
```

```
<!--include:C:/Users/Me/Docs/my_file.md-->
```

Multiple includes can be used in the same document.

```
<!--include:my_file.md-->


<!--include:doc_2.md-->
```

File Includes Behavior

When ePublisher detects a File Include statement, the file is read, and the Include tag is replaced with the content of the file. If the no file is found at the path given, the Include tag will be passed through to the output as an HTML Comment.

Using a File Include

To use an Include statement, all that needs to be done is write the tag where the file's content is to be imported. Below, an include statement is written below a Title.

```
Learning ePublisher

==================


<!--include:epublisher_basics.md-->
```

This can even be done inside of documents used in an Include statement, as long as it is not a [Recursive Include][mdp-includes-recursion]. Use this feature to create Map Files for many Markdown++ documents, or create documents needed for content re-use.

Recursive Includes

If an Include statement tries to insert a document that has already been inserted by a parent file, ePublisher's generation log will display a message like this one:

```
[Warning]

Skipping recursive include file:

 'C:\Users\Me\Documents\include_doc.md'

 in file: 'C:\Users\Me\main_doc.md'
```

This message displays because ePublisher cannot insert the document. Doing so would create a recursive loop and would

break the generation. If this message is recieved, it's time to look at the layout of Includes in the source documents.

The message can be useful to track down the file in error. The first file path refers to the file in the attempted Include statement. The second file path refers to where the Include occured.

Variables

Variables represent a shorthand to store a value that can be re-used across a set of documents. They're useful to store content that only needs to be written once, but is used in the same way in many places. Store values like product names, copyright text, publication dates, and more inside of Variables.

Syntax

Variables are the only syntax in Markdown++ that doesn't use the HTML Comment Tag.

To write one, start with `$`, write the variable name using alphanumeric characters, `-`, and `_`. End the Variable with `;`. Do not use spaces in the Variables's name.

Basics

A simple example of writing a Variable, called `product_name`.

```
$product_name;
```

Variables can be intermixed with other text content.

```
Document last published on $publish_date;.
```

The full range of Markdown features is available with Variables as well, both around them and written in their values.

```
The documentation for our product, **$product_name;**.
```

Variable Behavior

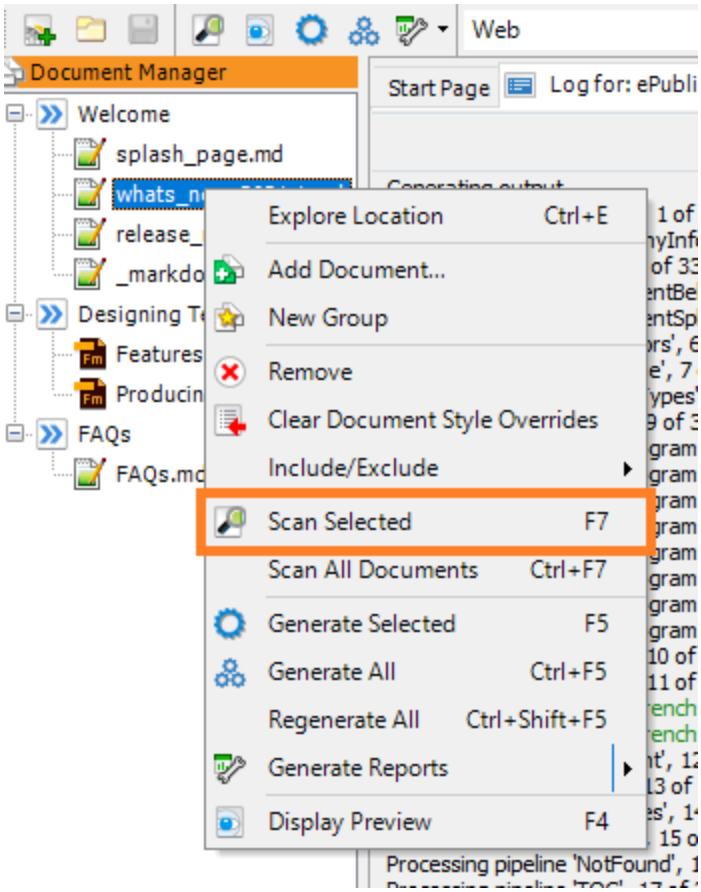
Variables, once scanned into ePublisher, can be given values that are saved to an ePublisher Project.

Using Variables

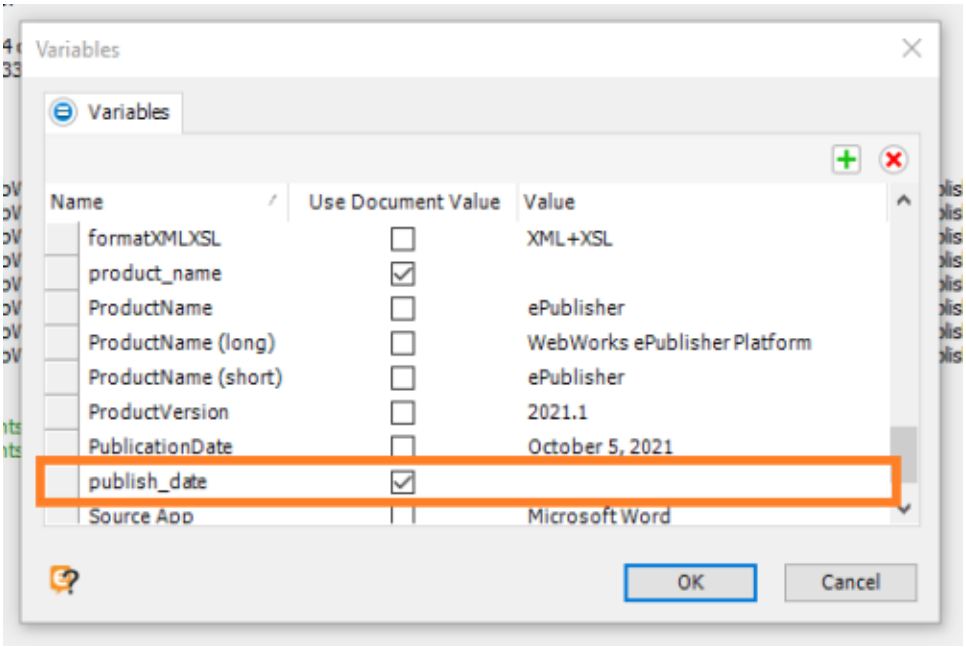
First, a Variable must be created by writing it into a document. Here, we create a Variable called `publish_date`.

```
Document last published on $publish_date;.
```

Next, scan the document in ePublisher. This will add the Variable to the Variables Window.



The Variable can now be given a value, typed in the input field next to the Variable's name in the Variable Window.



Use Document Value

The **Use Document Value** checkbox inside the Variables Window does not apply to Markdown++ Variables, since their values are instead maintained in ePublisher. There's no change in behavior based on if the box is checked or not. This feature applies to legacy source document types, such as FrameMaker and Word.

See Online Help for Markdown++ cheatsheet.