by **CLI Press**

Intro to ***Pressdown***

Generated: 2019-07-02 11:37:56

Table of Contents

Introduction to *Pressdown*

Pressdown is an extension of Markdown Extra that adds special processing directives to handle the specific needs of documentation tasks. Some of the directives help reduce markup, and others allow for easily escaping Twig, Markdown, and ***Pressdown*** directives. Most of all, though, ***Pressdown*** has lots of directives to make building your documentation easy and to make it beautiful.

This document provides detailed information and samples for all available ***Pressdown*** directives. You will see generalized versions of each directive as well as examples taken directly from the markup used to create this document. To make these easier to understand, these directives will be processed through a syntax highlighter. See ***Pressdown*** Syntax Highlighting for information on how to make sense of them.

Here is the ***Figure Directive*** in an abstract version as well as in a real-world version:

```
figure{content}(caption?)#anchor-name

figure{
Look I can use *Markdown* in here

Wow! I can use ^Pressdown^, too!
}(My caption can use *Markdown* and ^Pressdown^)#anchor-name
```

Each component of the directive will have a specific color that denotes how it is used.

`Purple`	Highlights the actual directive that lets ***Pressdown*** know how and what to parse.
`Blue`	A literal character.
`Off-white`	Denotes content that can contain Markdown and Pressdown.
`Light yellow`	Identifies options for the directive.

Note that in abstracts of a directive a `?` signifies that the preceding component is optional. So in the above, the `caption` component is optional, meaning the parentheses can be left empty.

Figure 1: ***Pressdown*** Syntax Highlighting

Pull Quotes

> **Grab the reader and draw them in**

The first directive we'll take a look at is the ***Pull Quote Directive*** - `pq{content}(float?anchor?)` . If you want to bring attention to an important phrase or idea on a page, you can use a pull quote to highlight it. They are often used to grab the reader and draw them in to a specific section, to prevent them from flipping past.

As you may have guessed, the component `content` should be the word or phrase you want to use. The `float` option specifies which way the pull quote will float wherever it appears. If not given, the default value is `right` . The `anchor` option can be the link name of a ***Pull Quote Anchor Directive*** that indicates where in the document flow the pull quote should appear. If it is not given, the pull quote will be anchored to wherever the original content is.

To use the ***Pull Quote Directive***, simply enter it wherever the phrase would normally occur in the flow of the document. The phrase will appear as normal in that location but also be displayed according to the `float` and `anchor` options. The example below shows how

the pull quote on this page is created.

```
They are often used to pq{grab the reader and draw them in}(left) to a specific section, to prevent them from flipping past
.
```

The pull quote on the previous page was located next to the original text. There's a pull quote on this page that is actually from a paragraph on the next page. Here is how this was accomplished:

```
// Here it is in the document flow
But as simple as the ^Classed Block Directive^ is, pq{ {{ pressDown }} has an even better way}(right better-way) to create
figures and captions.

// And here is how it is anchored in place
pqa{better-way}
```

Table 1: Summary of the *Pull Quote Directive* - `pq{content}(float? anchor?)`

Component	Description
`content`	This is what you want to appear, as is, within the normal flow of the document. It may contain Markdown and Pressdown.
`float`	This can be *left* or *right* and indicates which direction the pull quote should float. The default is *right* if no direction is given.
`anchor`	If an anchor name is provided it may only contain these characters: *A-Za-z0-9_-*. The pull quote will be placed immediately following the rendered pull quote `content`, or, if an anchor name is provided here, it will replace the linked ***Pull Quote Anchor Directive*** in the document flow.

Table 2: Summary of the *Pull Quote Anchor Directive* - `pqa{anchor-name}`

Component	Description
`anchor-name`	The anchor name used in a ***Pull Quote Directive*** elsewhere. The pull quote will be inserted into the document flow at the position of the directive.

The *Classed Block Directive*

One main feature is the ***Classed Block Directive*** - `{@tag-classcontenttag@}(options )?`. This allows you to briefly create HTML blocks with a specific assigned class. The `tag` can be replaced by a tag name or a one or two letter abbreviation that represents the following tags: *a*, *blockquote*, *code*, *figure*, *figcaption*, *div*, *li*, *ol*, *p*, *pre*, *span*, and *ul*. The `class` can be replaced by a single class name or a series of class names separated by a dot. You can also leave it empty and no class name will be applied to the block at all. The optional component `options` allows you to send special instructions to the Pressdown parser. See here for details. Figure A below shows a basic example.

> ***Pressdown* has an even better way to create figures**

With ***Pressdown*** this is how you make an element with a specific class:

```
// this
{@s-zI need to be a span with "z" classs@}

// becomes this
<span class="z">I need to be a span with "z" class</span>
```

Then you can use `.z` in your CSS files to do something cool.

Figure A: The ***Classed Block Directive*** `{@tag-class?contenttag@}`

You can nest classed blocks and use Markdown and Pressdown within them! (For now, you can only nest tags of different types, but nesting the same tag is under consideration as a feature.) Let's take a look at an example. Here is the Pressdown markup we used for Figure A. The Pressdown itself has syntax highlighting to help you better see the different components.

In the example markup below, `{{ pressDown }}` is a Twig expression for the press variable `pressDown`. Press variables aren't covered in this book, but you can find more information about them online. For now, just know that the press variable is a shortcut that itself translates into a ***Classed Block Directive***. That directive looks like this: `{@s-aPressdowns@}(-p)`. The ***Custom Directive*** will be covered a little later.

```
{@figure-With {{ pressDown }} this is how you make an element with a specific class:

custom{
// this:
{@s-zI need to be a span with "z" classs@}

// becomes this:
<span class="z">I need to be a span with "z" class</span>
}(code-block)

Then you can use `.z` in your CSS files to do something cool.figure@}
{@fc-Figure A: The ^Classed Block Directive^ {@tag-classcontenttag@}fc@}
```

After Twig, ***Pressdown***, and Markdown parsing, the final HTML markup looks like this:

```
<figure><p>With <span class="a">Pressdown</span> this is how you make an element with a specific class:</p>
<pre class="prettyprintplain"><p>// this:
<span class="z">I need to be a span with "z" class</span></p>
<p>// becomes this:
<span class="z">I need to be a span with "z" class</span></p></pre>
<p>Then you can use <code>.z</code> in your CSS files to do something cool.</p></figure>
<figcaption>Figure 1: The <em class="keyword1">Classed Block Directive</em> <code class="pressdown"><span class="pd-literal
">{@</span><span class="pd-directive">tag</span><span class="pd-literal">-</span><span class="pd-option">class</span><span
class="pd-pressdown">content</span><span class="pd-directive">tag</span><span class="pd-literal">@}</span></code></figcapti
on>
```

Table 3: Summary of the *Classed Block Directive* - `{@tag-class?contenttag@}(options)?`

Component	Description
`tag`	The HTML tag to use for the block. It can be either the full tag name or an abbreviation from this list: *a*, *blockquote*, *code*, *figure*, *figcaption*, *div*, *li*, *ol*, *p*, *pre*, *span*, and *ul*.
`class`	An optional class name to apply to the tag. You can specify multiple classes by separating them with a dot.
`content`	The contents of the class can be Markdown or Pressdown. You cannot nest another directive with the same tag.
`options`	This whole component is optional, i.e. you can leave off the parentheses entirely.

Figures and Captions

If you were paying attention above, you noticed that Figure A uses classed blocks of the `f(igure)` and `f(ig)c(aption)` variety. ***CLI Press*** automatically styles figures and captions to look like insets as the one above. But as simple as the ***Classed Block Directive*** is, ***Pressdown*** has an even better way to create figures and captions.

Introducing the ***Figure Directive*** - `figure{content}(caption?)#anchor-name`

Let's break down what that does, and how ***Pressdown*** helps you with it. The `content` is what will go in the `figure` tag. The `anchor-name` should be a unique name for the figure. This name allows you to create a link to the figure from any file using the ***Figure Link Directive***. Finally the `caption` is an optional caption that will be added to the bottom of the figure, like the caption in Figure A. And here's the best part: ***Pressdown*** will automatically number your figures for you, and prefix your caption with "Figure X:". Don't worry about renumbering figures. It's handled.

To see the directive in action, here's an example of how to create Figure A from the section on the ***Classed Block Directive*** using the ***Figure Directive***.

```
figure{
With {{ pressDown }} this is how you make an element with a specific class:

custom-2{
// this:
{@s-zI need to be a span with "z" classs@}

// becomes this:
<span class="z">I need to be a span with "z" class</span>
}(code-block)-2

Then you can use .z in your CSS files to do something cool.
}(The Classed Block Directive {@tag-class?contenttag@}(options)?)#CBD
```

And here's that Pressdown in action:

With ***Pressdown*** this is how you make an element with a specific class:

```
// this:
{@s-zI need to be a span with "z" classs@}

// becomes this:
<span class="z">I need to be a span with "z" class</span>
```

Then you can use `.z` in your CSS files to do something cool.

Figure 2: The *Classed Block Directive* `{@tag-class?contenttag@}(options)?`

Now suppose you want to refer to this figure later or earlier in the document. ***Pressdown*** can create a link for you that will automatically reference the proper number of the figure, regardless of where, or in what order, it was defined. Just use the name you provided in the ***Figure Directive*** and ***Pressdown*** will handle the rest. This magic is accomplished with the ***Figure Link Directive***: `f{caption?}(link)` . The `link` must match the anchor name in the ***Figure Directive***. The `caption` , if defined, will replace the default caption. For example `f{Classed Block Directive}(CBD)` , would create a link labeled "Classed Block Directive". If it were empty, the link would instead match the default caption, "The ***Classed Block Directive*** `{@tag-class?contenttag@}(options)?` ".

Table 4: Summary of the *Figure Directive* - `figure{content}(caption?)#anchor-name`

Component	Description
`content`	The content of the figure can be Markdown and Pressdown. In theory, you could even put a figure within a figure, but we would not recommend it.
`caption`	The optional caption can be Markdown and Pressdown. If provided, the caption will be prefixed with 'Figure X: ' where 'X' is an automatically tracked number. If empty, there will be no caption at all.
`anchor-name`	The anchor name must be unique across all ***Figure Directives*** and can be comprised of the following characters: *A-Za-z0-9_-*. It can be used in the ***Figure Link Directive*** to create clickable links to the figure.

Table 5: Summary of the *Figure Link Directive* - `f{caption?}(link)`

Component	Description
`caption`	If the caption isn't provided, the link text will default to the caption of the linked figure. If provided, it will be used in place of the default.
`link`	The link must be defined as an anchor name in a ***Figure Directive*** elsewhere.

Appendix I: *Pressdown* Directives Quick Reference

Table 6: *Pre Directives*

Directive Name	Directive Format	Purpose
Keyword One	`^content^`	Special ***emphasis***
Keyword Two	`^^content^^`	Special ***emphasis***
Keyword Three	`^^^content^^^`	Special ***emphasis***
Font Awesome	`{f@classes}`	Font Awesome icon
Page Break	`{break}`	Force page break

Table 7: *Block Directives*

Directive Name	Directive Format	Purpose
Classed Block	`{@tag-class?contenttag@}(options)?`	HTML block with tag and optional class. More than one class may be specified by separating them with dots. The `options` component sends processing instructions to the Pressdown parser. A list of options can be found here.
Pull Quote	`pq{content}(float? anchor?)`	Aside with quoted phrase. Optional components `float` and `anchor` designate how and where to place the pull quote.
Alert	`alert{content}(icon class?)`	Inset text box with a configurable Font Awesome icon and optional class

Table	`table-cols{`
	`content`
	`}(caption?)#anchor-name`
		Simple table with optional caption

Figure	`figure{`
	`content`
	`}(caption?)#anchor-name`
		Inset figure with optional caption

Custom	`custom-level?{`
	`content`
	`}(directive  options?)-level?`
		Custom directive defined in a theme or project configuration. They can be nested with the use of the optional component `-level` where `level` is a single digit, so up to ten directives can be nested.

Table 8: *Post Directives*

Directive Name	Directive Format	Purpose
Figure Link	`f{caption?}(link)`	Link to named figure with optional replacement caption
Pull Quote Anchor	`pqa{name}`	Location for pull quote defined elsewhere
Table Link	`t{caption?}(link)`	Link to named table with optional replacement caption

Table 9: *Final Directives*

Directive Name	Directive Format	Purpose
Escaped Code Blocks	``@````	Prevents Markdown parsing
Escaped Twig Expression	`{@{ expression }}`	Prevents Twig parsing

Appendix II: Options for *Pressdown* Directives

Table 10: Options for the *Classed Block Directive*

Option	Description
`-final`	This will prevent further processing of the block. Some Pressdown processing will occur even with this option.
`-md`	This will prevent further Markdown processing of the block.
`-nw`	This will prevent all wrapping of the block. Good for keeping code snippets in one line.
`-p`	This will strip all Markdown `p` tags from the block using this regex: `/<\/?p>/` .
`+w`	The default action is to trim whitespace from the contents. This will prevent that from happening.

Table 11: Options for the *Custom Directive*

Option	Description
`-final`	This will prevent further processing of the block. Some Pressdown processing will occur even with this option.
`-p` and `+p`	This will strip/not strip all Markdown `p` tags from the block using this regex: `/<\/?p>/` .
`-p2br` and `+p2br`	This will convert/not convert all Markdown `p` tags in the block to `br` tags.