

THE KNOCKOUT TUTORIAL

“ using microsoft word

moves an image a mm to the left

all text and images shift. four new pages appear. paragraph breaks form a union. a swarm of commas buzzes at the window. in the distance, sirens.”

— laurelhach, February 3, 2016
321,101 notes (as of June 10, 2016)

Like Laurel over there, I too got fed up with Word’s sh-t.

Much to my disappointment, I could not find a real alternative to Word in the free software community. LibreOffice was modeled on Word, and so the same flaws that make Word so annoying to use make LibreOffice even more annoying to use. LaTeX was too unwieldy and foreign and didn’t seem like much fun to use. Inkscape, while flexible, was utterly inappropriate for setting orderly passages of text. So I exercised the right and duty of any FOSS citizen worth his salt—I made my own alternative.

“ D’you like it?”

— Taylor Swift, *I Wish You Would* voice memo

1 HOW DOES KNOCKOUT WORK?

Knockout is a low-level, class-based, modular **typesetting engine** and editor with hybrid user interaction. A typesetting engine is simply something that translates textual information (and styling information that goes with it) into a visual layout. The input of a typesetting image is text and data, like the string `'astounding'` or something more

sophisticated like a graph with a point `'P'` at the coordinate `(3, 4)`. The typesetting engine then performs something called **layout**, when it decides where on the page everything should go. Finally, it does something called **rendering**, which actually draws the letters and shapes. When the output of rendering is expressed in pixels, this process is known as **rasterization**.

2 DOCUMENT CLASSES

Knockout styles documents using **classes**, not **direct styling**. Direct styling “bakes” styling information into the text. Class-based styling, on the other hand, tags text according to its meaning (“heading”, “body text”, “caption”, etc) and uses that to compute the styling of the text. This is useful because it separates meaning from styling in a document. For example, you can edit how you want the `'h2'` class to look, and the change will propagate to every paragraph that uses `'h2'`. Much easier than going to every second-level heading in your document and manually changing the styling!

2.1 Block-level styling

Block-level styling refers to styling that affects how layout is done on paragraphs (and other modules). Knockout contains the following block-level styling attributes:

`leading`
`float [=22]`

The height of each line of text, in pixels. The baseline of the text is anchored to the bottom edge of each leading strip. Leading in Knockout is always constant (there is no spreading of lines).

This paragraph has a `leading` property value of `10` pixels.

`align`
`float [=0]`

The distribution of the space on either side of the text on each line. `0` puts all the space on the right (flush left). `1` puts all the space on the left (flush right). `0.5` puts half the space on the left and half on the right (flush center). As you might guess, `0.389` puts 38.9% of the space on the left and 61.1% of the space on the right.

This line has `align = 0.2`

This line has `align = 0.5`

This line has `align = 0.8`

This line has `align = 1`

`align on`
`string [=']`

The characters that Knockout will attempt to anchor the line on. The characters are ordered in decreasing power. If `align on` is not empty, the tab character `'\t'` is implied to be ranked first. Thus, with the value `'.: -'`, Knockout will first search the line for a tab character; if there is none, it will search for a `'.'`, then it will search for a `':'`, and finally it will search for a `'-'`. If there are still no matches, Knockout will place the anchor at the end of the line. This is very useful for aligning tables of numbers.

```

1989,131.6209
1,989,131.62
  31,228.1248589
    403.293.590
      22:13:56
22:13:56.04
  45-a1-78-BB
    45-a1:78-BB
      45-a1:78.BB
45,a1,78,BB
  45,a1-78.BB

```

`indent`
`binomial [=0']`

The amount that indented lines are pushed inward, in the form `a + bK`, where `a` is in pixels, and `b` is the number of characters `K` the line is indented.

—» The `K` term, when used with a negative coefficient, is useful for creating hanging indents like this one.

The `indent` property should not be used to typeset numbered or bulleted lists; use `incr_place_value`, `incr_assign`, `show_count`, and `counter_space` instead.

`indent_range`
`integer set [=0']`

The indices of the lines in each paragraph that the `indent` is applied to. For example, a value of `0, 2` indents the first and third lines of the affected paragraphs, such as this one (used with `indent = 20`). This property is labeled as **for lines** in the Knockout UI.

`margin_left`
`float [=0]`

The left margin of the block element, in pixels. This property is labeled as **space left** in the Knockout UI.

This paragraph has a `margin_left` value of `60` pixels.

`margin_right`
`float [=0]`

The right margin of the block element, in pixels. This property is labeled as **space right** in the Knockout UI.

This paragraph has a `margin_right` value of `60` pixels.

`margin_top`
float [=0]

The upper margin of the block element, in pixels. This property is labeled as **space before** in the Knockout UI. This property only takes effect between two blocks; it is irrelevant for the first block in any frame.

This paragraph has a `margin_top` value of 60 pixels.

`margin_bottom`
float [=0]

The lower margin of the block element, in pixels. This property is labeled as **space after** in the Knockout UI. This property only takes effect between two blocks; it is irrelevant for the last block in any frame.

This paragraph has a `margin_bottom` value of 60 pixels.

`hyphenate`
boolean [=False]

Whether or not hyphenation is performed on the paragraph.

`keep_together`
boolean [=False]

Whether or not to treat the block element as inseparable over multiple frames. If set to `True`, the block element will not split over frame breaks. It's strongly recommended to set this property to `True` on graphs and charts.

`keep_with_next`
boolean [=False]

Whether or not to move the block element to the next frame if the following block element breaks into the next frame. It's recommended to set this property to `True` for headings. Overuse of this property may make editing chaotic as it forces Knockout to re-layout preceding block elements to a limited degree. The names and types of all property documentations in this manual have `keep_with_next` set to `True`.

`incr_place_value`
integer [=1]

The place value in the Knockout block element counter that `incr_assign` affects. Can be any integer from negative infinity to 12, inclusive. This property is labeled as **increment** in the Knockout UI.

`incr_assign`
function (n) : integer [=None]

Returns the new value of the Knockout counter at `incr_place_value`, with the old value `n` as input. Setting this property to `'n + 1'` will increment the counter by 1. Setting this property to `'None'` will disable incrementation. If `incr_place_value` is positive, `incr_assign` will also overwrite all counter place values of higher index with 0. This property is labeled as **by** in the Knockout UI.

`show_counter`
function (A) = string [=None]

Returns the string to be rendered alongside the block element, with the array `A` as input. For example, `"'. '.join(A[:4])"` will print the first four place values of the Knockout counter, separated by periods. `'None'` will disable counter printing. This property is labeled as **counter text** in the Knockout UI.

`counter_space`

`float [=0.5]`

The distance, as a factor of the paragraph `leading` value, that the printed counter is spaced to the left of the leftmost edge of the block element.

- [X] This counter is has a `counter_space` value of `0.5`.
- [X] This counter is has a `counter_space` value of `1`.
- [X] This counter is has a `counter_space` value of `1.5`.