Logical CSS

If you've been following the CSS specifications you will see that several of the more recent ones have addopted `start` and end as values for the properties and I wondered why was this, wasn't it enough to work with top, bottom, left and right?

It wasn't until I started working with and learning about writing modes that the reasoning became clear. when working from top to bottom and left to right then it's easier to think about the layout in the way we normally do, it also saves us from code duplication by eliminating code that we'd use to handle right-to-left code when the default is left-to-right.

According to caniuse.com only Firefox has full support for the properties while Chrome and Safari (and Chrome for Android, UC Browser for Android and Samsung Internet) have partial support behind the `webkit` flag.

Edge has the feature under consideration and high priority according to the Edge Platfom Status entry.

PostCSS To The Rescue

Since we can't be sure if our target browsers support the specific logical features that we want, we can use PostCSS' logical CSS plugin to get the behavior we want now, rather than waiting for browsers to catch up.

I've added the `postcssLogical` plugin to my processCSS task.

```
gulp.task('processCSS', () => {
  // What processors/plugins to use with PostCSS
  const PROCESSORS = [
    postcssLogical({ dir: 'ltr' }),
    autoprefixer({ browsers: ['last 3 versions'] })
  ];
  return gulp'ltr'.src('src/css/**/*.css')
    .pipe($.sourcemaps.init())
    .pipe(postcss(PROCESSORS))
    .pipe($.sourcemaps.write('.'))
    .pipe(gulp.dest('src/css'))
```

```
    .pipe(
      $.size({
        pretty: true,
        title: 'pr'src/css/**/*.css'
    );
});
```

Then take the following CSS that uses inset and padding-inline and run it through the processor.

the `inset` property defines the logical block offset values of an element, which maps to a physical offset depending on the element's writing mode, directionality, and text orientation. It may correspons to the top, right, bottom, or left property depending on the values defined for writing-mode, direction, and text-orientation.

In this case, when using three values the first value corresponds to `top`, the second one to `left` and `right` and the third one to `bottom`.

```
.banner {
  color: #222222;
  inset: logical 0 5px 10px;
  padding-inline: 20px 40px;
  resize: block;
  transition: color 200ms;
}
```

The result will be like follows:

```
.banner {
  color: #222222;
  top: 0;
  left: 5px;
  bottom: 10px;
  right: 5px;

  &:dir(ltr) {
    padding-left: 20px;
```

```
    padding-right: 40px;
  }

  &:dir(rtl) {
    padding-right: 20px;
    padding-left: 40px;
  }

  resize: vertical;
  transition: color 200ms;
}
```

Notice that we got two additional selectors, one to handle left to right (`:dir(ltr)`) and one for right to left (`:dir(rtl)`) for free. While this is essential for right-to-left writing modes, it's also fun to have when experimenting with writing modes as a creative tool.

Perhaps my favorite logical property is `block-size`. It defines the horizontal or vertical size of an element's block, depending on its writing mode. It corresponds to either the width or the height property, depending on the value of writing-mode.

This little beauty will let me work with text in vertical and horizontal layouts without having to remember whether I'm working with height or width. Win!

Links and Resources

- CSS Logical Properties
  - Spec
  - MDN Docs
- Logical CSS PostCSS Module
  - Githubb Repo
  - Logical CSS PostCSS Module README
- Writing
  - CSS Grid, Logical Values and Writing Modes
  - CSS Writing Modes
  - Understanding Logical Properties And Values