Building a Website with Make4ht: Styling with CSS

David Friant

May 9th, 2026

Abstract

As has been alluded to many times in the preceeding articles, CSS is heavily used to ensure a clean and consistent style across the website while also providing certain functionalities which would be very difficult to achieve in plain HTML. Style-wise, colors and fonts are discussed. Functionality-wise, changing color modes, tool-tip behavior, syntax highlighting, code line numbering, and image scaling are all discussed.

Key Words: LaTeX, Make4ht, Styling, CSS, IndieWeb

Introduction

As has been heavily alluded to by the previous articles in this series, the CSS styling[1] of this website is relied upon for a few key features. The most important of all these are the fonts, colors, navigation menu, table of contents, and the handling of the display of all those in both desktop and mobile formats. Furthermore, there are a few specific features that have should receive specific attention as they have been referenced in previous articles: changing the color mode dynamically, tool-tip like behavior, code coloring and automatic line numbering, image scaling, and horizontal scrollbars for content with a width greater than that of the normal text column. As a final note before starting, this article will not cover every single definition part of the CSS file in detail. Instead, small snippets will be extracted for discussion. Reader who wish to peruse the full file may find it using their web browser's dev tools.

Fonts and Colors

Naturally, the first point of discussion should be the most obvious visual characteristics of the website: fonts and colors. Listing 1 provides the definitions of a number of CSS variables for both the font families to use in the document and what sizes to use in various parts of the website. Of these, special mention should be made of the font icons. These reference a custom web font[5] created to contain the glyphs[6] for the special icons used in the navigation menu and a few other places such as the link icon which appears when hovering over a section title.

Readers will also note the "duplicate" font size definitions. These represent the font sizes to use on either a desktop or mobile screen, though discussion of their use will wait until the section dedicated to just that. Further, one may also not that while sans-serif and monospace fonts are used a great deal on this website, serif fonts are used extremely sparingly. The definition of the `--font-text-serif` variable is thus seemingly unnecessary, however it is wise to future-proof in this case as it is not unreasonable that some content may one day use it.

Listing 1: The basic definitions for various fonts and their sizes to be called in when setting things for mobile and desktop environments.

```
/*Custom Font Definition*/
@font-face{
        font-family: FriantIcons;
        src: url(icons.woff2)
```

```
}

:root{
        /*Fonts*/
        --font-text-serif:      'Times New Roman', Times, serif;
        --font-text-sans-serif:  Arial, Helvetica, sans-serif;
        --font-text-monospace:  'Courier New', Courier, monospace;
        --font-icons:            FriantIcons;

        --font-size-desktop-normal-text:   15pt;
        --font-size-desktop-small-text:    12pt;
        --font-size-desktop-h1:            30pt;
        --font-size-desktop-h2:            25pt;
        --font-size-desktop-h3:            20pt;
        --font-size-desktop-h4:            15pt;
        --font-size-desktop-h5:             8pt;
        --font-size-desktop-h6:             4pt;
        --font-size-desktop-nav-head:      30pt;
        --font-size-desktop-nav-icon:      15pt;
        --font-size-desktop-nav-body:      15pt;
        --font-size-desktop-linkicon:      17pt;

        --font-size-mobile-normal-text:  3.50vw;
        --font-size-mobile-small-text:   1.74vw;
        --font-size-mobile-h1:           5.00vw;
        --font-size-mobile-h2:           4.07vw;
        --font-size-mobile-h3:           3.49vw;
        --font-size-mobile-h4:           2.91vw;
        --font-size-mobile-h5:           2.33vw;
        --font-size-mobile-h6:           1.74vw;
        --font-size-mobile-nav-head:    10.00vw;
        --font-size-mobile-nav-icon:     8.00vw;
        --font-size-mobile-nav-body:     6.00vw;
        --font-size-mobile-linkicon:     5.00vw;
}

.custom-icons{
        font-family: var(--font-icons);
}
```

As was the case with fonts, Listing 2 presents the definitions for most of the colors that are used on this website. While the variables in Listing 1 varied depending on the size of the screen, these vary depending on the color-mode of the website. Broadly speaking, the user's OS and/or browser can choose either the light or dark color scheme (the first or second parameter of the `light-dark(l,d)` function). Thus, the colors should be acceptable on the user's first visit to the website. Discussion of manually changing the color mode and saving that change are reserved for a dedicated section below.

Special note should be made of the data series color variables. As was mentioned in the previous article, these must be kept in close accord with the LaTeXcolor definitions and the mapping from those colors to the various variable names in the post-processing script.

Listing 2: The basic definitions of the for the colors used on this website for both light and dark modes.

```
:root{
    /*Use the OS/Browser set light or dark color scheme*/
    color-scheme: light dark;

    /*General Colors*/
    --text:                   light-dark(rgb( 40,  40,  40),
                                         rgb(230, 230, 230));
```

```
    --text-visited-link:      light-dark(rgb( 40,  40,  40),
                                         rgb(230, 230, 230));
    --text-unvisted-link:     light-dark(rgb( 40,  40,  40),
                                         rgb(230, 230, 230));
    --background:             light-dark(rgb(230, 230, 230),
                                         rgb( 40,  40,  40));
    --text-select:            light-dark(rgb(230, 230, 230),
                                         rgb( 40,  40,  40));
    --text-select-background: light-dark(rgb( 40,  40,  40),
                                         rgb(230, 230, 230));
    --text-highlight:         light-dark(rgb(135, 135, 135),
                                         rgb(135, 135, 135));
    --text-hover:             light-dark(rgb(135, 135, 135),
                                         rgb(135, 135, 135));
    --nav-directory-hover:    light-dark(rgb(128, 128, 128),
                                         rgb(128, 128, 128));
    --nav-directory-active:   light-dark(rgb(185, 185, 185),
                                         rgb( 85,  85,  85));
    --border:                 light-dark(rgb(135, 135, 135),
                                         rgb(135, 135, 135));
    --primary-accent:         light-dark(rgb(183, 183, 183),
                                         rgb( 87,  87,  87));
    --secondary-accent:       light-dark(rgb( 87,  87,  87),
                                         rgb(183, 183, 183));

    /*Data Series Colors: Okabe and Ito*/
    --data-series-oai-0:      light-dark(rgb( 40,  40,  40), /*Text, Light   */
                                         rgb(230, 230, 230));/*Text, Dark    */
    --data-series-oai-1:      light-dark(rgb(  0, 158, 115), /*Bluish Green  */
                                         rgb(  0, 158, 115));
    --data-series-oai-2:      light-dark(rgb(  0, 114, 178), /*Blue         */
                                         rgb(  0, 114, 178));
    --data-series-oai-3:      light-dark(rgb( 86, 180, 233), /*Sky Blue     */
                                         rgb( 86, 180, 233));
    --data-series-oai-4:      light-dark(rgb(210, 198,  36), /*Yellow, Light */
                                         rgb(220, 208,  46));/*Yellow, Dark  */
    --data-series-oai-5:      light-dark(rgb(230, 159,   0), /*Orange       */
                                         rgb(230, 159,   0));
    --data-series-oai-6:      light-dark(rgb(213,  94,   0), /*Vermillion   */
                                         rgb(213,  94,   0));
    --data-series-oai-7:      light-dark(rgb(204, 121, 167), /*Reddish Purple*/
                                         rgb(204, 121, 167));
}
```

Menu and Table of Contents

One of the smaller, yet most useful features is the ability to lock certain elements to fixed positions on the screen and make them collapsible. This is the case for the navigation menu on the left side of the screen and the table of contents menu on the right. Listings 3 and 4 provide a minimalist example of HTML and CSS documents which, together, allow this effect.

Listing 3: A minimalist HTML example to demonstrate collapsing menus.

```
<!DOCTYPE html>
<html>
<head>
        <title >Menu Example</title>
        <link rel="stylesheet" href="Menus.css">
</head>
<body>
```

```
<nav id="menu">
        <div class="menu-button" onclick="collapse(this.parentElement)">
                Button
        </div>
        <div class="menu-body">
                Menu element to collapse.
        </div>
</nav>
<script>
/*Collapse a collapsible element such as a menu*/
function collapse(elem){
        if(elem.className.includes(' active')){
                elem.className = elem.className.replace(' active', '');
        }
        else{
                mode = document.documentElement
                elem.className = elem.className + ' active';
        }
}
</script>
</body>
</html>
```

On the HTML side of the system, the elements are arranged such that the parent element [A] contains a button which may be clicked to change the collapse state and other content which is to be collapsed out of sight when the button is clicked. Note that the button itself must not be collapsed lest the state become irreversible. The functional part of this is the definition and call of the `collapse(elem)` method. This is called when the button is clicked and passes the `<nav>` element as the method's argument.

The javascript function itself is defined in the `<script>` block. Here, the function simply checks if the element passed to it is of the class `active`. If yes, the class is removed. If not, the class is added to the element. This allows a simple toggle of the class of the element which may then be queried by the CSS as seen in Listing 4.

Listing 4: A minimalist CSS example to demonstrate collapsing menus and fixed positions.

```
/*Nav Menu*/
nav{
        position: fixed;
        top: 0;
        left: 0;
        height: fit-content;
}

nav .menu-button{
        background-color: gray;
        color: white;
}

nav .menu-body{
        display: none;
}

nav.active .menu-body{
        display: block;
        position: relative;
}
```

The CSS necessary for the base functionality for this collapsing behavior is quite minimal, as understood from Listing 4. Simply, the non-button child elements have two possible selectors to provide them styles, one if the parent element is of the `active` class and one if it is not. In the case it is not, the `display` property is set to `none`, thus preventing the element and all of its children from being drawn. In the other case, the `display` property is set to `block`, thus ensuring that it is drawn as a block element.

One should also make note of some of the properties defined for the `<nav>` element itself. The position is fixed and locked to the top-left corner of the screen. This ensures that the menu stays on screen at all times regardless of where the webpage is scrolled to. Further, its `height` property is set to `fit-content`. This ensures that the element will always expand in height as much as it needs to in order to contain the elements within it. This changes dynamically as sub-elements are collapsed and expanded.

[A] The `nav` element in this case.

Considerations for Desktop and Mobile

As mentioned above, the design considerations for desktop and mobile viewing experiences are different. In fact they are sufficiently different that many large websites have dedicated subdomains for handling traffic for mobile users with entirely different CSS. However, this is not and never will be a website with enough visitors for that to be a consideration. As such, the goal is to change as little as possible in the single unified CSS file for the website. To that end, media queries[4] can be used to change CSS properties depending on the size of the user's screen.

Listing 5 provides the CSS to do just that. Note that this is a fairly rough cut and doesn't really account for intermediately sized devices such as some tablets which are bigger than a phone but smaller than a typical desktop. Note also that the media queries set variables with the same names, thus allowing other elements to use those as the values of properties regardless of size of the screen. This layer of abstraction allows all of the changes to be made in one place instead of spread throughout the entire CSS document. This also, ultimately, reduces the size of of the CSS file for the same reason. One may consider the color definitions to behave in the same manner, i.e. defining the universally-used variable in a different manner in a context-dependent way.

Listing 5: CSS media queries to set font sizes and other metrics to attempt to provide a clean experience across both mobile and desktop devices.

```
@media only screen and (max-width: 920px){
    :root{
        --font-size-normal-text:    var(--font-size-mobile-normal-text);
        --font-size-small-text:     var(--font-size-mobile-small-text);
        --font-size-h1:             var(--font-size-mobile-h1);
        --font-size-h2:             var(--font-size-mobile-h2);
        --font-size-h3:             var(--font-size-mobile-h3);
        --font-size-h4:             var(--font-size-mobile-h4);
        --font-size-h5:             var(--font-size-mobile-h5);
        --font-size-h6:             var(--font-size-mobile-h6);
        --font-size-nav-head:       var(--font-size-mobile-nav-head);
        --font-size-nav-icon:       var(--font-size-mobile-nav-icon);
        --font-size-nav-body:       var(--font-size-mobile-nav-body);
        --font-size-linkicon:       var(--font-size-mobile-linkicon);

        --nav-max-width:            var(--mobile-nav-max-width);
        --nav-margin:               var(--mobile-nav-margin);
        --nav-directory-radius:     var(--mobile-nav-directory-radius);
        --nav-directory-step-width: var(--mobile-nav-directory-step-width);
        --content-max-width:        var(--mobile-content-max-width);
    }
}

@media only screen and (min-width: 920px){
    :root{
        --font-size-normal-text: var(--font-size-desktop-normal-text);
        --font-size-small-text:  var(--font-size-desktop-small-text);
        --font-size-h1:          var(--font-size-desktop-h1);
        --font-size-h2:          var(--font-size-desktop-h2);
        --font-size-h3:          var(--font-size-desktop-h3);
        --font-size-h4:          var(--font-size-desktop-h4);
        --font-size-h5:          var(--font-size-desktop-h5);
        --font-size-h6:          var(--font-size-desktop-h6);
        --font-size-nav-head:    var(--font-size-desktop-nav-head);
        --font-size-nav-icon:    var(--font-size-desktop-nav-icon);
        --font-size-nav-body:    var(--font-size-desktop-nav-body);
        --font-size-linkicon:    var(--font-size-desktop-linkicon);

        --nav-max-width:            var(--desktop-nav-max-width);
```

```
        --nav-margin:               var(--desktop-nav-margin);
        --nav-directory-radius:     var(--desktop-nav-directory-radius);
        --nav-directory-step-width: var(--desktop-nav-directory-step-width);
        --content-max-width:        var(--desktop-content-max-width);
    }
}
```

Specific Features

There are a number of features that require special mention as they've been referenced by other articles in this series. These may be found below.

Changing Color Modes

The `light-dark()` function[2] is a built-in way to define the colors for both a light mode and dark mode for an element in a single place. This is convenient as it reduces the amount of code that needs to be written and maintained. By default, this uses the color scheme provided by the reader's operating system or browser. However, it is possible (and highly desirable!) to give readers the option to manually change the color mode by a simple button click.

Listing 6 provides a simple demonstration of the HTML side of precisely that. Here the button is set up to trigger the `swapColorMode()` function when clicked. This function simply checks if the color scheme is set to "light" and swaps it to "dark" if so. Otherwise the color scheme is set to "dark". One should note that the color scheme is actually set twice here, once to `localStorage` and once to the document's dataset[3]. Setting the color scheme in the `localStorage` allows it to be retrieved after closing the website, browser, or even computer, while setting the dataset value allows it to be accessed cleanly from CSS.

Listing 6: A basic HTML document set up to demonstrate the groundwork necessary to use the `light-dark()` function by button press.

```
<!DOCTYPE html>
<html>
<head>
    <title >Menu Example</title>
    <link rel="stylesheet" href="LightDark.css">
</head>
<body>
<main>
    <div>
        <button type="button" onclick="swapColorMode()">Swap Colors</button>
    </div>
    <div>
        This is some test text.
    </div>
</main>
<script>
function swapColorMode(){
    mode = document.documentElement.dataset.colorScheme;
    if(mode == "light"){
        document.documentElement.dataset.colorScheme = "dark";
        localStorage.setItem("color-scheme", "dark");
    }
    else{
        document.documentElement.dataset.colorScheme = "light";
        localStorage.setItem("color-scheme", "light");
    }
}
</script>
</body>
</html>
```

Listing 7: An example of the basic CSS machinery necessary to allow a user to change color scheme at their discretion.

```
:root{
    /*Modes*/
    color-scheme: light dark;
    &[data-color-scheme="light"] {
        color-scheme: light;
        #lightModeIcon{
            display: none;
        }
        #darkModeIcon{
            display: inline-block;
        }
    }
    &[data-color-scheme="dark"] {
        color-scheme: dark;
        #lightModeIcon{
            display: inline-block;
        }
        #darkModeIcon{
            display: none;
        }
    }
}

body{
    background-color: light-dark(
        rgb(var(--background-light)),
        rgb(var(--background-dark)));
    color: light-dark(
        rgb(var(--text-light)),
        rgb(var(--text-dark)));
}

body ::selection{
    background-color: light-dark(
        rgb(var(--text-select-background-light)),
        rgb(var(--text-select-background-dark)));
    color: light-dark(
        rgb(var(--text-select-light)),
        rgb(var(--text-select-dark)));
}
```

Listing 7 provides the rest of the code necessary to allow users to change the color scheme at their discretion. Here, the CSS code in the `:root{}` selection first sets the color scheme to allow both light and dark modes. Then the dataset is queried for the color scheme. The colors are set appropriately and the relevant buttons are hidden or revealed such that there is always a button press to change the color mode, but the button itself changes. This allows the swapping of the graphic for the color mode as can be seen on the HTML version of this document. The rest of Listing 7 simply provides some basic `light-dark()` function definitions to demonstrate how to do so. The two code listings function correctly if copied into appropriately named files and opened in a browser.

Tool-tip Behavior

To finish the discussion on tool-tips that was started in the third article of this series, Listing 8 provides the CSS necessary to invoke the effect from the HTML elements created there. Broadly, the CSS simply ensures that the tool-tip content is not drawn by default. It is drawn only when the tool-tip mark (typically a number or capital Latin letter) is hovered over with the mouse or, importantly, when the tool-tip content itself is hovered over. This allows clickable content such as links to be embedded in the tool-tip and accessed without vanishing again when the reader moves to click on it.

Listing 8: The CSS necessary to get tool-tip behavior for footnotes and citation.

```css
/*Tooltips*/
span.tooltipcontainer{
    position: relative;
}

sup.tooltipmark a{
    font-size: var(--font-size-small-text) !important;
    position: relative;
    z-index: 0;
}

span.tooltip{
    display: none;
    position: absolute;

    background-color: var(--nav-directory-active);
    opacity: 0;
    transition: opacity var(--hover-mode-transition-duration);

    border: 1px solid var(--border);
    border-radius: 10px;
    padding: 5px;

    width: max-content;
    max-width: 50vw;
    top: 0;
    left: 0;
    text-align: justify;
    z-index: 2;
}

sup:hover + span.tooltip{
    display: block;
    position: absolute;
    opacity: 0;
    z-index: 2;
}

span.tooltip:hover{
    display: block;
    position: absolute;
    opacity: 1;
    z-index: 2;
}
```

Listing 9: Some JavaScript to ensure that tool-tip content stays on screen without being cut off.

```javascript
/*Keep absolute positioned elements on the screen.*/
function keepOnScreen(elem){
    //X direction
    var rect = elem.getBoundingClientRect();
    if(window.innerWidth < rect.x + rect.width + 5){
        elem.style.left = window.innerWidth - rect.x - rect.width - 5 + "px";
    }

    //Y Direction
```

```
    var r = document.querySelector(':root');
    var f = getComputedStyle(r).getPropertyValue('--font-size-normal-text');
    var p = getComputedStyle(r).getPropertyValue('--padding-width');
    var b = getComputedStyle(r).getPropertyValue('--border-width');
    var footHeight =
        Number(f.substring(0, f.length - 2)) +
        2 * Number(p.substring(0, p.length - 2)) +
        Number(b.substring(0, b.length - 2));
    if(window.innerHeight < rect.y + rect.height + footHeight){
        elem.style.top =
            window.innerHeight - rect.y - rect.height - footHeight - 5 + "px";
    }
}

/*Reset positions after when stop hovering*/
function resetPositions(elem){
    elem.style.left = "0px";
    elem.style.top = "0px";
}
```

An unfortunate side effect of defining tool-tips in the fashion described above is that they can be drawn only partially on the screen, e.g. a tool-tip on the right side of the articles content may be draw with its right-most content cut off by the screen edge. This can be remedied with the scripts provided in Listing 9. Conceptually, the `keepOnScreen(elem)` function simply checks if any part of the tooltip is off the right and bottom sides of the screen. If yes, the position is adjusted such that the content is entirely kept on screen. The `resetPositions(elem)` function resets the position of the tool-tip once it is no longer being hovered over such that the effect of repeatedly hovering over the tool-tip that may not stack and thus maneuver the content off screen.

Code Coloring and Automatic Line Numbering

Code, both inline and in block listings, is one of the easier things to style after going through the work detailed in the article dedicated to code highlighting. The CSS provided by Listing 10 dictates that inline code snippets, e.g. `void myFunction(int arg)`, are not colored but they are surrounded by a dashed box to ensure that they stand out while not overly breaking the flow of the text.

Code listings are rather more complicated, but still straightforward. Readers should first note the definition of the CSS counter `line` which is initialized with a value of -1. This is used to track the line count and automatically generate line numbers for each line of the code block. This is brought about by incrementing the `line` counter for each `<code>` element that is encountered within the `<pre>` block which defines a code listing. The numbers themselves are drawn by way of the `content: counter(line)` definition in the part of the CSS file which defines things that happen before the `<code>` elements. As a beneficial side effect of this process, the line numbers are not "real" in the sense that they do not exist in the HTML document. As such, the code blocks may be copied from without fear of copying the line numbers.

Listing 10: The CSS required to style code, both inline and in block listings.

```
/*Generic inline code*/
code{
        border: 1px dashed var(--border);
        font-family: var(--font-text-monospace);
        padding-left: var(--padding-width);
        padding-right: var(--padding-width);
        transition: color var(--color-scheme-transition-duration) ease,
                                border var(--color-scheme-transition-duration) ease;
}

/*Generic Code Block*/
pre{
        border: 1px solid var(--border);
        font-family: var(--font-text-monospace);
```

```
  font-size: var(--font-size-small-text);
line-height: 1.2em;
  transition: color var(--color-scheme-transition-duration) ease,
              border var(--color-scheme-transition-duration) ease;
counter-reset: line -1;

& code{
        display: block;
        border: none;
        margin: 1px;
        counter-increment: line;
          transition: color var(--color-scheme-transition-duration) ease,
                border var(--color-scheme-transition-duration) ease;

        & .w{/*Whitespace*/
                color: var(--text);
        }
        & .err{/*Error*/
                color: var(--token-unhandled);
                transition: inherit;
        }
        & .x{/*???*/
                color: var(--token-other);
                transition: inherit;
        }
        & .k{/*Keyword*/
                color: var(--token-keyword);
                transition: inherit;
        }
        /*
         * ...
         * Many Lines of color definitions supressed for brevity
         * ...
         */
        & .c1{/*Comment.Single*/
                color: var(--token-comment);
                transition: inherit;
        }
        & .cs{/*Comment.Special*/
                color: var(--token-comment);
                transition: inherit;
        }
}

/*Zebra stripes*/
& code:nth-child(even){
        background-color: light-dark(
                rgba(  40,  40,  40, 0.04),
                rgba( 230, 230, 230, 0.04));
}

/*Line counter*/
& code::before{
        display: inline-block;
        border-right: 1px solid var(--border);
        width: 5%;
        padding-right: 0.5em;
        margin-right: 0.5em;
```

```
                text-align: right;
                content: counter(line)
        }
}
```

As far as highlighting the code goes, the colors corresponding to each token type are simply assigned one after the other as seen in Listing 10. The colors themselves are defined in a manner identical to that seen in Listing 2. Slightly more interesting is the zebra striping. This uses the `::nth-child(even)` selector to ensure that every other line of code has a slightly different colored background to ease the eye in tracking left to right across the screen. This is an old, but still very useful technique for reducing strain on readers who may be reading a great deal of intricately detailed code. One could also imagine using a hover selector to highlight a given line of code, but that has not been implemented here.

Image Scaling

One of the smaller, yet highly important features of CSS is the ability to control the scale of images. The CSS styling in Listing 11 sets the minimum and maximum widths for both raster and vector images, effectively clamping their sizes to be between 50 and 100% of the width of the text column. This ensures that the images remain clear and legible regardless of their actual size[B]. The `margin-left: auto;` and `margin-right: auto;` ensure that the images are centered if they do not take up the full width of the text column. Note also the `image-rendering: crisp-edges;` statement; this ensures that raster images that are very small (e.g. Figure 1 of the article on images) are drawn such that each pixel is clearly visible without any blending or interpolation.

Listing 11: The CSS styling for images, both raster and vector.

```
/*Figures*/
div.figure{
        width: 100%;
        margin-top: 1em;
        margin-bottom: 2em;

        position: relative;
        z-index: -1;
}

img{
        max-width: 100%;
        min-width: 50%;
          display: block;
          margin-left: auto;
          margin-right: auto;
        image-rendering: crisp-edges;
}

svg{
        max-width: 100%;
        min-width: 50%;
          display: block;
          margin-left: auto;
          margin-right: auto;
}
```

[B]Within what is possible by scaling the image, of course. Text which is small in comparison to the image size may be too small to read when scaled down to fit

Horizontal Scrollbars

The final and smallest CSS detail to be discussed here is the addition of horizontal scroll bars to overly-wide elements. As the only elements which are not somehow scaled to fit within the text column are tables, they are the only ones to be discussed. Quite simply, adding the `overflow-x: auto` to the `<div class="tabular">` elements ensures that the content contained within them will never expand beyond their width. That width, of course, being the width of the text column. In the event that the content's width is smaller than than the `<div>`, no scroll bar is shown. In the event that it is larger, a horizontal scroll bar is shown such that the reader can access the full image, though not all at once.

Listing 12: The CSS styling to ensure that overly-large tables are given horizontal scroll bars instead of rendering outside of the bounds of the text column.

```
/*Tables*/
div.tabular{
        width: 100%;
        max-width: 100%;
        overflow-x: auto;
}
```

The Next Article

This concludes (for now) everything that should be said about using LaTeXand Make4ht to produce HTML documents for use on the web. The next article in this series is tentatively its last. There one may find a discussion on the tooling designed to generate not just a single web page, but an entire website of (mostly) static HTML.

References

[1] Mdn web docs: Cascading style sheets. `https://developer.mozilla.org/en-US/docs/Web/CSS`. Accessed: 2026-04-19.

[2] Mdn web docs: Color scheme. `https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/color_value/light-dark`. Accessed: 2026-04-19.

[3] Mdn web docs: Dataset property. `https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset`. Accessed: 2026-04-19.

[4] Mdn web docs: Media queries. `https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Media_queries/Using`. Accessed: 2026-04-19.

[5] Mdn web docs: Web fonts. `https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Text_styling/Web_fonts`. Accessed: 2026-04-19.

[6] Wikipedia: Glyphs. `https://en.wikipedia.org/wiki/Glyph`. Accessed: 2026-04-19.