# CSS containers: making websites accessible to disabled users

John R Hudson

1 May 2020*

## 1 Personal background

I first became interested in page design as a student journalist at university. When I first started using a micro-computer, I was interested in how much time it could save me; when I discovered vector graphics in the mid 1980s, I realised that I could save time and produce quality output by using a computer. In the 1990s I worked with a blind student and learned how I could use my computing skills to make life easier for him.

After 30 years of using my understanding to create quality printed documents and more recently PDF files, in 2010 I was inspired by a talk from David Fisher to put these principles to work in maintaining and developing websites.

## 2 The wider context

Cascading style sheets (CSS), proposed in 1994 by Håkon Wium Lie, who worked at CERN, were first adopted by Microsoft in Internet Explorer 3 and then by Netscape and Opera. These separate the presentation of material on a website from its content and structure as defined by HTML.

At the turn of the century, they went out of favour for a number of reasons and development focused on XHTML. But, a few years later, Mozilla, Apple and Opera began working on a new specification for HTML which would meet the needs of modern devices and which would rely on CSS for presentation. This was published in 2011 as a rolling release and work began on a series of rolling updates to CSS to cope with the needs of the new version of HTML and the demands of modern devices.

## 3 The semantic elements, HTML and CSS

As part of their work, Mozilla, Apple and Opera surveyed a large number of existing websites and concluded that five things, later to be known as the semantic elements, featured in most web pages: a header, a footer, a navigation bar, material which was tangential to the main content and the main content (figure 1).

*Originally intended for presentation at the openSUSE Summit Dublin.

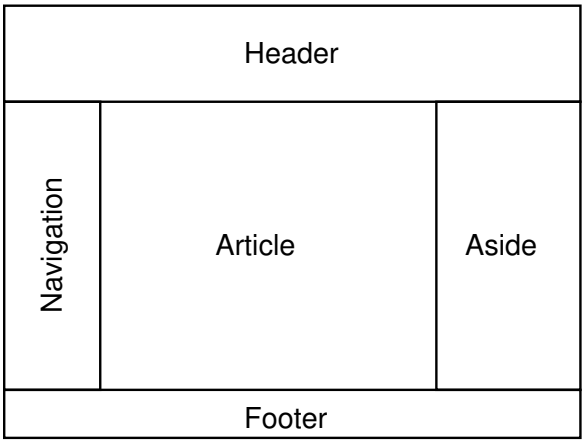


Figure 1: The five semantic elements (thanks to David Fisher (2010))

During the first decade of the century, screens had moved from being close to a square to being an oblong as first laptops and then smartphones grew in popularity. Putting the navigation bar and any tangential content on either side of the main content made sense because it reduced the distance which the reader's eye had to jump back to the start of the next line in the main content.

Mozilla, Apple and Opera went through all the existing elements which they found in various implementations of HTML, obsoleting those which had a presentational function and, as far as possible, redefining the functions of those where there was some ambiguity about their function in terms of their content rather than any presentational aspect they might have.

The redefinition of HTML elements made HTML far more readable and easy to maintain than earlier versions of HTML or XHTML. It also needed less code to achieve many things, thereby reducing the burden on browsers and the delay for users accessing a website, and the same code could be used to generate a website on a smartphone and on a large monitor.

## 4 Web Content Accessibility and the ARIA attributes

In 2008 the Web Content Accessibility Guidelines were published by W3C and work began on the development of the WAI-ARIA attributes; these were formally accepted as a W3C Recommendation in 2014 and continue to be developed.

In addition, in the redefinition of HTML elements, a number have useful accessibility functions. For example, the `<time>` element allows a machine readable date and time to be recorded alongside a human readable form while the `<i>` element has a `lang=""` attribute enabling you to specify the language of the text in the `<i>` element so that a screenreader knows that it has to change language if it can.

However, I have yet to come across any website generation software which fully implements the ARIA attributes as most rely on obsolete HTML to accomplish what can be achieved more simply by using the redefined HTML elements and CSS.


Figure 2: Using a CSS flex container

## 5 Using the semantic elements

One of the significant weaknesses of the redefined HTML when it was first developed was any means of accurately placing elements on a page; for example, though you were recommended to put the `<nav>` and `<aside>` elements on either side of the `<article>` element (figure 1), there was no easy way to ensure reasonably secure placement of these elements on the page.

Using `nav {float: left;}` and `aside {float: right;}` in the CSS file ensured that the `<nav>` and `<aside>` elements were on either side of the `<article>` element but, if they were of significantly different lengths, the `<footer>` element would move left or right to be under the shorter elements.

It also meant that, as when navigation bars are placed at the top of a web page, a screenreader user still had to read through the entire navigation bar before reaching the content of the page.[1]

I was initially puzzled by the appearance of the `<main>` element in the HTML standard but all became clear with the arrival of CSS containers. Containers can be `flex` or `grid` containers. A `flex` container expands or contracts the elements within it horizontally or vertically so that they fit the available width or height. A `grid` container allows you to organise elements of uneven dimensions in rows and columns to provide the most suitable disposition of elements.

Figure 2 from the CSS Flexible Box Layout specification demonstrates the canonical use of CSS containers to improve accessibility for screenreader users. Within the HTML code, the `<article>` element containing the main content of the page, precedes the `<nav>` and `<aside>` elements so that a screenreader user can read the main content of the page first. But all three elements are children of a `<main>` element which is defined in the CSS file as a `flex` container.

[1] Rian Rietveld recommends adding a link such as `<a class="skip-link screen-reader-text" href="#content">Skip to content</a>` before the navigation links and specifying the height of this link in the CSS file as 1px. This makes the text too small for sighted users to see in a browser but it is displayed in a screenreader so that a screenreader user can select it to jump to the main content. So far I have found only a few websites which take advantage of this, some of which mess up the link anyway!

```
body {font: normal 100% serif;}
header {border-top: medium solid #C71585; border-right: medium solid #C71585; border-left: medium solid #C71585;
background: transparent url("graphics/Topback_new.png") no-repeat; padding-right: 1em;}
h1 {font: bold 180% sans-serif; text-align: right; color: gold;}
h2 {font: bold 160% sans-serif; text-align: right; color: gold;}
h3 {font: bold 150% sans-serif; text-align: center; color: #C71585;}
h4 {font: bold 120% sans-serif; color: #C71585;}
h5 {font: bold 100% sans-serif; color: #C71585;}
main {display: flex;}
main>article {order: 2;  width: 58%; padding-left: 3em; padding-right: 3em;}
main>nav {order: 1; width: 19%; padding: 1em;}
main>aside {order: 3; width: 19%; padding: 1em;}
section.grid {display: grid; grid-template-columns: auto auto; justify-content: space-around;}
*.footer {font: normal 80% sans-serif; text-align: center;}
p.first {font: normal 120% serif;}
p.right {font: italic 100% serif; text-align: right;}
li.nav {font: bold 120% cursive; list-style: none; line-height: 150%}
li.aside {font: bold 100% sans-serif; list-style: none; line-height: 150%}
li.alpha {list-style: lower-alpha;}
li.roman {list-style: lower-roman;}
td {padding-right: 1em; vertical-align: top;}
figure.left {float: left;}
figure.right {float: right;}
figcaption {text-align: center; font-style: italic;}
span {color: #C71585;}
sub, sup {font-size: 60%;}
```

Figure 3: A CSS file

The `order` property in the CSS file specifies the order in which each element will appear on the screen; so the `<nav>` element appears first — on the left — for sighted readers, followed by the `<article>` element and then the `<aside>` element.

## 6 An example of a CSS file in use

The CSS file from the Heath Old Boys Association documentation (figure 3) illustrates the use of both a `flex` and a `grid` container in this static website.

Heath Old Boys Association is an association of former pupils of Heath Grammar School, Halifax, which closed in 1985 through a merger with another school in the town. As the former pupils of the school will increasingly be older people who might require accessibility aids, the website was redesigned to use the redefined HTML elements and the ARIA attributes some years ago and CSS containers were implemented in 2019.

```
main {display: flex;}
```

declares that the `<main>` element acts as a CSS `flex` container.

The subsequent rules specify that, when a particular element is a child of the `<main>` element, its properties will take the values specified.

```
main>article {order: 2; width: 58%; padding-right: 3em;
padding-left: 3em;}
main>nav {order: 1; width: 19%; padding: 1em;}
main>aside {order: 3; width: 19%; padding: 1em;}
```

Because several pages of the website consist of lists — for example, the Obituaries page consists of a list of obituaries — it was helpful for sighted readers to create two columns on this and other pages containing lists. As these elements would be children of the `<article>` element, a `<section>` element acting as a CSS `grid` container was specified in the following rule:

```
section.grid {display: grid; grid-template-columns: auto auto;
justify-content: space-around;}
```

Though at the moment I cannot see another use for a `<section>` element in the website, I decided to give it the `class="grid"` attribute for future proofing.

`grid-template-columns` allows you to specify a space separated list of columns; so `grid-template-columns: auto auto;` specifies two automatically sized columns.

`justify-content: space-around;` distributes items evenly along the horizontal axis with a half space at either end.

The CSS Grid Layout allows for much more complex placing of elements than the CSS Flexible Box Layout — asymmetrically, with items of different heights and widths, with names for items and with the possibility of changing the layout to suit different devices and orientations. The CSS Flexible Box Layout changes automatically to suit different devices and layouts but only in one dimension, not two like the CSS Grid Layout.

In the HTML code I have used `<section class="grid">` to introduce the container and then a `<div>` element for each column. A screenreader presents the two `<div>` elements as a single list with a small gap half way down.

I have not yet had a reason to test out a complex CSS grid container and how it affects the way a screenreader presents the content but I suspect that some care will need to be taken in developing grid containers which may be helpful to sighted users to ensure that they do not make the experience worse for visually impaired users.

## 7 Conclusion

With the arrival of CSS containers, the project initiated by Mozilla, Apple and Opera fifteen years ago has finally come to fruition, providing a range of features which surpasses anything available in earlier editions of HTML or XHTML. For example, the CSS `voice-` properties have been completely revised so that, if you have dialogue, quotations or verse in a page, you can specify the type of voice the screenreader will use, male or female, young or old, as well as the pitch, the speed, the volume and so on.

There is no reason now for hanging on to the old ways of developing websites and there is the very good reason of making the Internet accessible to everyone for adopting CSS containers and all the other improvements to CSS which make websites more accessible to those with disabilities.

## References

Fisher, D. (2010, 25 August). HTML5. Talk at the Bradford GNU/LUG meeting at Bradford Council for Voluntary Services, Bradford, West Yorkshire.