Articles » Web Development » HTML / CSS » General

# Fun Exploring Div and Table Layout

Marc Clifton

12 Aug 2019 CPOL

Comparing and contrasting UI layout using divs vs. tables.

**Download Examples.zip - 2.4 KB**

## Contents

- Introduction
- Basic Layout
  - Float vs. Inline Block
  - Reading Between the Lines
    - Lesson 1 - Your editor is messing with your head
    - Lesson 2 - Your minimizer will mess with your head too
    - Lesson 3 - All the solutions are bad
- Left, Center, and Right, Atten-hup!
  - Lesson 3 - floats, not inline-block
  - Lesson 4 - Always Test Different Widths
  - Vertical Element Positioning
  - Clearing Float and Inline Block
  - Fixed Width Left and Right Divs
  - Div Scrolling
- Using Tables instead of Divs
  - Lesson 5 - Hidden Spacing
  - Fixed Width Columns
  - Column Scrolling
- Conclusion

## Introduction

This is a straightforward article demonstrating some very basic UI layout concepts using either DIV or TABLE elements.  The impetus for this is simply that I often struggle with figuring out the nuances of how to position elements in the UI so I get both the look and behavior that I want.  By look, I mean the positioning of sections of the UI and the elements within those sections, and by behavior, I mean how the UI behaves when the browser window is resized.  I'm not covering things like device screen sizes or replacing a menu bar with a "triple bar" dropdown -- there are component libraries like Bootstrap to handle that.

All of these examples can be coded in Visual Code and viewed side-by-side with Quick HTML Previewer, which of all the HTML preview plugins that I looked at is the only one that I found actually works.

## Basic Layout

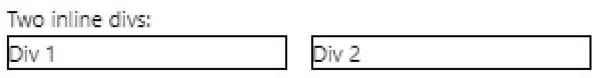
Let's start with something basic -- two inline divs.  The HTML:

```
<p>Two inline divs:</p>
<div class='border1 quarter inline'>Div 1</div>
<div class='border1 quarter inline m10'>Div 2</div>
```

And the CSS I'm using to define the border, width, margin, and inline styles:

```
.border1 {
  border-style:solid;
  border-width: 1px;
}

.quarter {
  width: 25%;
}

.inline {
  display: inline-block;
}

.m10 {
  margin-left:10px;
}
```

Yielding:



### Float vs. Inline Block

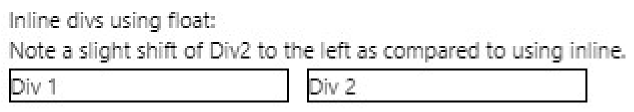
If you want some in depth reading about the two, I suggest Ternstyle's blog entry and Thoughtbot's blog entry.  For now, let's just deal with the practical issues, otherwise known as WTF???

Let's compare that with using a `float` style instead.  The HTML:

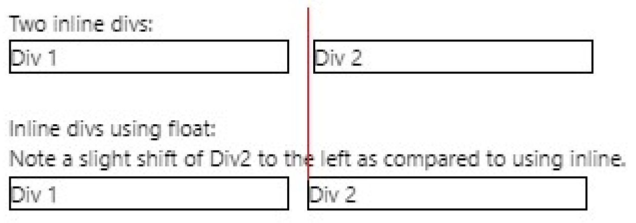
```
<p>Inline divs using float:<br>
Note a slight shift of Div2 to the left as compared to using inline.</p>
<div class='border1 quarter fleft'>Div 1</div>
<div class='border1 quarter fleft m10'>Div 2</div>
```

And the `fleft` CSS:

```
.fleft {
  float:left;
}
```


Inline divs using float:
Note a slight shift of Div2 to the left as compared to using inline.
Div 1 Div 2

Looks pretty much the same, right?  But notice there's a slight spacing difference between the two divs:

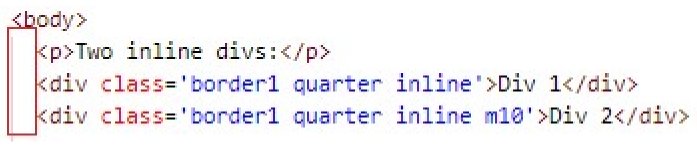

Two inline divs:
Div 1 Div 2

Inline divs using float:
Note a slight shift of Div2 to the left as compared to using inline.
Div 1 Div 2

Why is that?  Well, read on.

## Reading Between the Lines

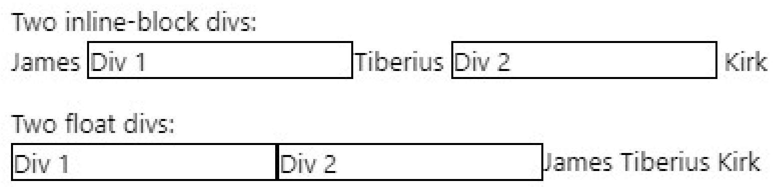
As it turns out, using `inline-block` respects any text between the divs, whereas `float` moves the text between the divs out to the right.  This is an important behavioral difference!  So the difference between the divs using `inline-block` is because I have spaces in the HTML:


```
<body>
  <p>Two inline divs:</p>
  <div class='border1 quarter inline'>Div 1</div>
  <div class='border1 quarter inline m10'>Div 2</div>
```

See the indentation of my nicely formatted HTML?  Let's exaggerate this difference between `inline-block` and `float`, getting rid of my left margin and removing the editor's indentation:

```
<p>Two inline-block divs:</p>
James <div class='border1 quarter inline'>Div 1</div>Tiberius
<div class='border1 quarter inline'>Div 2</div> Kirk

<p>Two float divs:</p>
James <div class='border1 quarter fleft'>Div 1</div>Tiberius
<div class='border1 quarter fleft'>Div 2</div> Kirk
```

And we get:

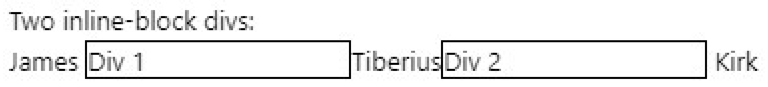

Two inline-block divs:
James Div 1 Tiberius Div 2 Kirk

Two float divs:
Div 1 Div 2 James Tiberius Kirk

Oh wow.  Even the carriage return between the `inline-block` divs is adding a space:

```
<p>Two inline-block divs:</p>
James <div class='border1 quarter inline'>Div 1</div>Tiberius<div class='border1 quarter
inline'>Div 2</div> Kirk
```

Notice the space before the second div has now been removed:


Two inline-block divs:
James Div 1 TiberiusDiv 2 Kirk

### Lesson 1 - Your editor is messing with your head

You're lovely HTML indentation, and in particular, the *auto-indenting* that your editor is doing for you, is affecting your layout using `inline-block`!

### Lesson 2 - Your minimizer will mess with your head too

So you've created the perfect layout in your editor and then you run a minimizer which potentially removes the whitespace and carriage returns between your divs.  Suddenly you UI looks slightly different!

### Lesson 3 - All the solutions are bad

All of the solutions are ugly.  From here and here, they are:

- Remove all whitespace and carriage returns between divs.  So much for readable HTML.
- Use float, but that has side-effects like we saw above with text between the divs.
- Create a container div with `font-size` of 0 then override the font size in the child divs.  Gross.

Any my favorite driving-the-porcelain-bus solution:

```
James<!--
--><div class='border1 quarter inline'>Div 1</div>Tiberius<!--
--><div class='border1 quarter inline'>Div 2</div> Kirk
```

OK then.  So there really is, at least in my opinion, no viable solution that maintains a nicely formatted HTML document and renders the same identical layout when the HTML is minimized.  Which basically means, always test your layout as minimized HTML rather than in a preview editor.  And of course there's no preview editor that I know of that will minimize your HTML before feeding it to the browser.  Hmmm, there's another article idea!  Of course, if you have a really smart minimizer that respects whitespace and carriage returns between and around `inline-block` divs, that would help too.  Anyone with better suggestions?

## Left, Center, and Right, Atten-hup!

Most of the time I need layout options that include being able to position an entire section to the left, center, and/or right of the page as well as position the elements within those sections on the left, center, right and top, middle, or bottom. That's not too much to ask for, is it?  Riiiight.

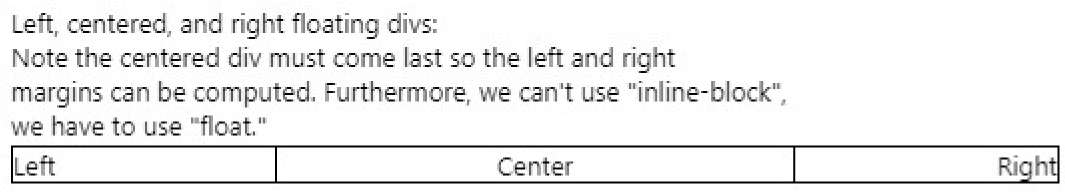
Here's the HTML for a simple layout (note I'm using `float`):

```
<p>Left, centered, and right floating divs:<br>
Note the centered div must come last so the left and right<br>
margins can be computed. Furthermore, we can't use "inline-block",<br>
we have to use "float."</p>
<div class='border1 fleft quarter'>Left</div>
<div class='border1 fright quarter taright'>Right</div>
<div class='border1 center tacenter'>Center</div>
```

and the additional CSS:

```
.center {
  margin-left:auto;
  margin-right:auto;
}

.quarter {
  width: 25%;
}

.fleft {
  float:left;
  padding: 0;
  margin: 0;
}

.fright {
  float:right;
}

.tacenter {
  text-align: center;
}

.taright {
  text-align: right;
}
```

And the result:


Left, centered, and right floating divs:
Note the centered div must come last so the left and right
margins can be computed. Furthermore, we can't use "inline-block",
we have to use "float."

| Left | Center | Right |
|---|---|---|

## Lesson 3 - floats, not inline-block

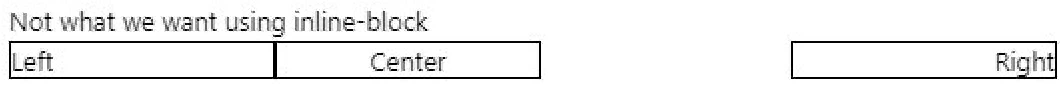
- We have to use `float`!

- The center div has to come last!

If we try to use `inline-block` with float, the center auto-margin is ignored.  Using this HTML:

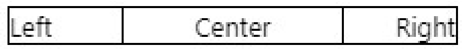
```
<div class='border1 inline fleft quarter'>Left</div>
<div class='border1 inline center quarter tacenter'>Center</div>
<div class='border1 inline fright quarter taright'>Right</div>
```

we get:

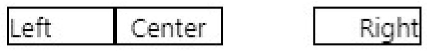

Not what we want using inline-block

The problem here is that the we're specifying the width of the center region.  This means that the center doesn't dynamically expand the way the center float version does when the browser width shrinks.

Using float:



Using inline-block:



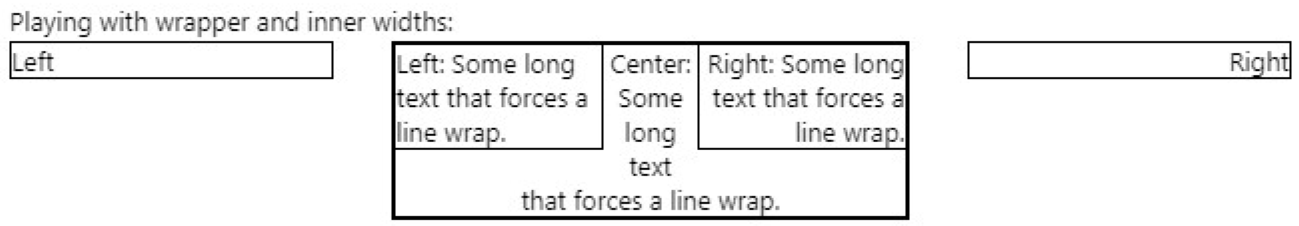
Instead, the center div width, using inline-block, is determined by the content of the div.  This makes it impossible to right-justify text against the left edge of the rightmost div.  Got that?  If there is a solution, I haven't found it.  Anyone?

## Lesson 4 - Always Test Different Widths

We can get some bizarre effects.  Given this HTML:

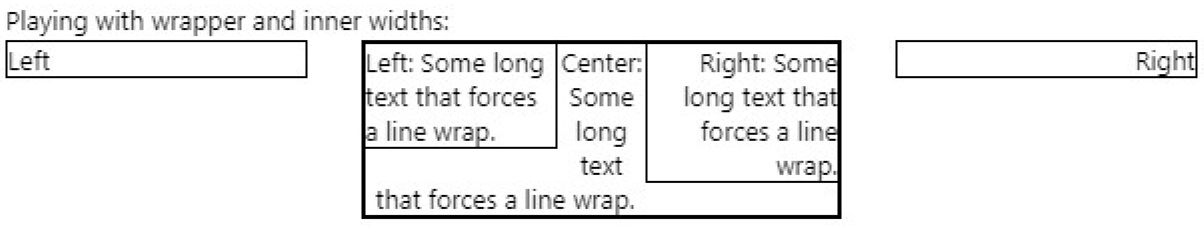
```
<p>Playing with wrapper and inner widths:</p>
<div class='border1 fleft quarter'>Left</div>
<div class='border1 fright right quarter taright'>Right</div>
<div class='border1 center tacenter w40p'>
  <div class='border1 fleft taleft w40p'>Left: Some long text that forces a line
wrap.</div>
  <div class='border1 fright right taright w40p'>Right: Some long text that forces a line
wrap.</div>
  <div class='border1 center tacenter'>Center: Some long text that forces a line
wrap.</div>
</div>
```

We can resize the width of the browser window and we see:


Playing with wrapper and inner widths:

Left
Left: Some long text that forces a line wrap.
Center: Some long text that forces a line wrap.
Right: Some long text that forces a line wrap.
Right

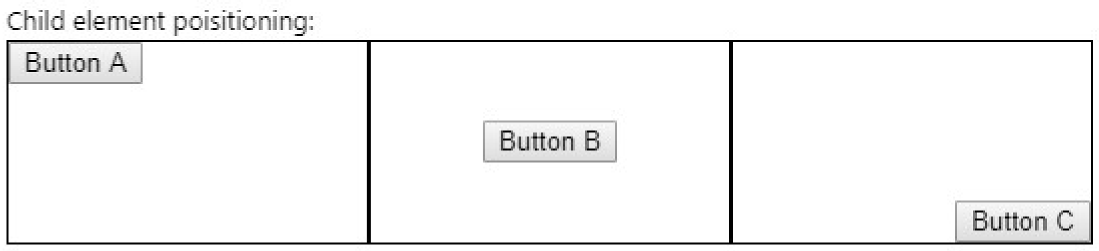
But wait!  A little different width and we get:


Playing with wrapper and inner widths:
Left
Left: Some long text that forces a line wrap.
Center: Some long text that forces a line wrap.
Right: Some long text that forces a line wrap.
Right

This effect is the result of the inner right div's height forcing the inner-center text at the bottom to not be able to extend the full width of the center div.  Makes sense, right?  The point being, always test your layout with different browser widths.  Within reason, of course!

## Vertical Element Positioning

I discovered vanseo design's blog post that solved this problem for me, so what you see here is just a regurgitation of their post.  To achieve this:



Requires treating the divs as table cells!  Here's the HTML:

```
<p>Child element poisitioning:</p>
  <div class='parent border1 fleft w33p h100px'>
  <div class="child-top-left"><button>Button A</button></div>
</div>
  <div class='parent border1 fleft w33p h100px'>
  <div class="child-middle-center"><button>Button B</button></div>
</div>
  <div class='parent border1 fleft w33p h100px'>
  <div class="child-bottom-right"><button>Button C</button></div>
</div>
```

and here's the additional CSS:

```
.w33p {
  width: 33%;
}

.h100px {
  height: 100px;
}

.parent {display: table;}

.child-top-left {
  display: table-cell;
}

.child-middle-center {
  display: table-cell;
  vertical-align: middle;
  text-align: center;
}

.child-bottom-right {
  display: table-cell;
  vertical-align: bottom;
  text-align: right;
}
```

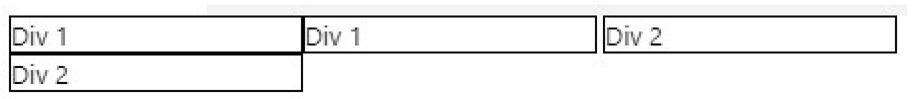
Funny how we use the `text-align` style to align inner HTML elements!  So this works because we're creating divs that simulate table cells, which is a natural lead in to the next section, doing the same thing with tables.  But first...

## Clearing Float and Inline Block

Consider this fragment:

```
<div class='border1 quarter inline'>Div 1</div>
<div class='border1 quarter inline m10'>Div 2</div>

<div class='border1 quarter fleft'>Div 1</div>
<div class='border1 quarter fleft m10'>Div 2</div>
```

Resulting in:



That isn't what we want at all!  To deal with this, we need to clear the float elements on the left and right:.

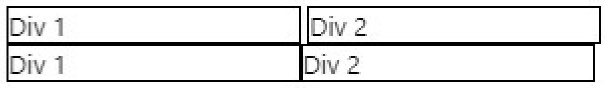
HTML:

```
<div class='border1 quarter inline'>Div 1</div>
<div class='border1 quarter inline m10'>Div 2</div>

<div class='clear'></div>

<div class='border1 quarter fleft'>Div 1</div>
<div class='border1 quarter fleft m10'>Div 2</div>
```

CSS:

```
.clear {
  clear:both;
}
```

Resulting in a "new line":

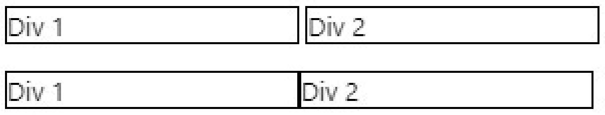


While a paragraph creates the same effect, it adds undesirable vertical separation between the two divs.

HTML:

```html
<div class='border1 quarter inline'>Div 1</div>
<div class='border1 quarter inline m10'>Div 2</div>

<p></p>

<div class='border1 quarter fleft'>Div 1</div>
<div class='border1 quarter fleft m10'>Div 2</div>
```

Result:



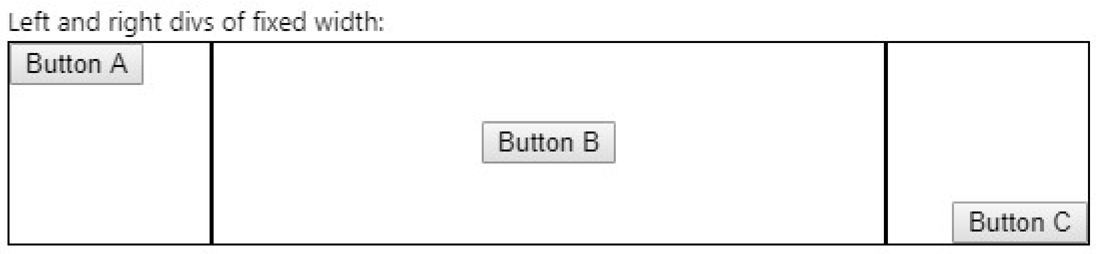
## Fixed Width Left and Right Divs

Let's say you want 3 divs where the left and right are of fixed (in pixels) width and the center div resizes based on the browser width. The only solution I found (after perusing SO) is to use the calc style:

```html
<p>Left and Right divs of fixed width:</p>

<div class='parent border1 fleft w100px h100px'>
  <div class="child-top-left"><button>Button A</button></div>
</div>
<div class='parent border1 fleft h100px' style='width: calc(100% - 206px);'>
  <div class="child-middle-center"><button>Button B</button></div>
</div>
<div class='parent border1 fleft w100px h100px'>
  <div class="child-bottom-right"><button>Button C</button></div>
</div>
```

Since this calculation is so dependent on the number of divs in the "row", I decided to put this in as an actual `style` rather than as CSS. The number 206 comes from the left and right divs being 100 pixels each, plus 1 pixel for the borders for each div: 100 + 100 + 1 + 1 + 1 + 1 + 1 + 1 = 206.

The result works nicely regardless of browser window width:


Left and right divs of fixed width:

Not ideal with this magic number, and compare this to the solution using tables.

According to the Mozilla docs: "The calc() CSS function lets you perform calculations when specifying CSS property values. It can be used anywhere a <length>, <frequency>, <angle>, <time>, <percentage>, <number>, or <integer> is allowed."

## Div Scrolling

A lot of times you'll see a div that has a scrollbar when the content exceeds the height.  Let's see how that's done using the overflow-y and height styles.

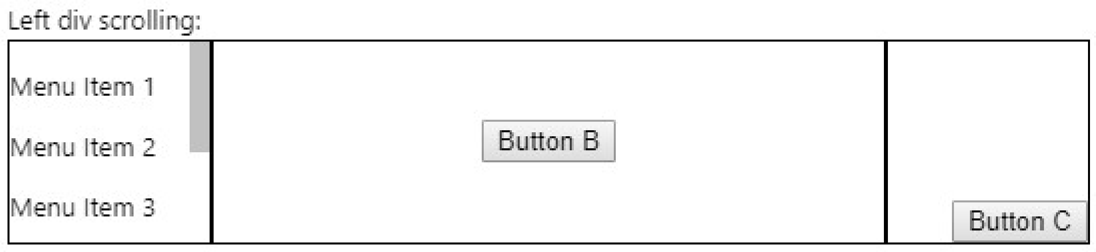
HTML:

```
<p>Left div scrolling:</p>
<div class='parent border1 fleft w100px h100px'>
  <div class="child-top-left">
    <div class='scrolly h100p'>
      <p>Menu Item 1</p>
      <p>Menu Item 2</p>
      <p>Menu Item 3</p>
      <p>Menu Item 4</p>
      <p>Menu Item 5</p>
      <p>Menu Item 6</p>
    </div>
  </div>
</div>
  <div class='parent border1 fleft h100px' style='width: calc(100% - 206px);'>
  <div class="child-middle-center"><button>Button B</button></div>
</div>
  <div class='parent border1 fleft w100px h100px'>
  <div class="child-bottom-right"><button>Button C</button></div>
</div>
```

Additional CSS:

```
.h100p {
  height: 100%;
}

.scrolly {
  overflow-y: auto;
}
```

Note that we have to use an inner div for the child div.  The result is:


Left div scrolling:

Menu Item 1

Menu Item 2

Menu Item 3

The scrollbar goes away if the height of the child's inner div is less than the child div.  What we lose though is the ability to vertically align the inner elements; they can still be horizontally aligned.  This makes a kind of sense, as why would you have a scrollbar with content that is vertically centered or at the bottom?

## Using Tables instead of Divs

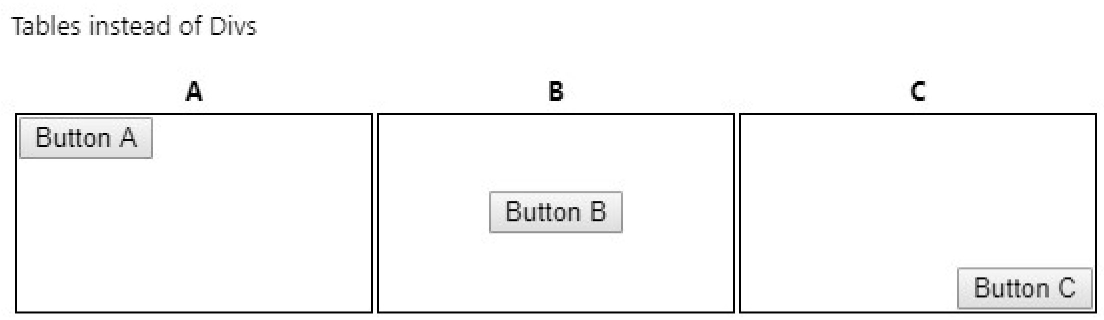
Let's cut right to the chase and reproduce the previous layout using actual tables, rows, and columns.  Here's the HTML:

```
<p>Tables instead of Divs</p>
<table class='w100p'>
  <tr>
    <th>A</th>
    <th>B</th>
    <th>C</th>
  </tr>
  <tr class='h100px'>
    <td class='border1 w33p vatop'><button>Button A</button></td>
    <td class='border1 w33p tacenter vacenter'><button>Button B</button></td>
    <td class='border1 w33p taright vabottom'><button>Button C</button></td>
  </tr>
</table>
```

And the CSS:

```
.w100p {
  width: 100%;
}

.taleft {
  text-align: left;
}

.tacenter {
  text-align: center;
}

.taright {
  text-align: right;
}

.vatop {
  vertical-align: top;
}

.vamiddle {
  vertical-align: middle;
}

.vabottom {
  vertical-align: bottom;
}
```

Resulting in:



Notice the space between the columns?  To get rid of that, we have to use the `border-collapse` style:

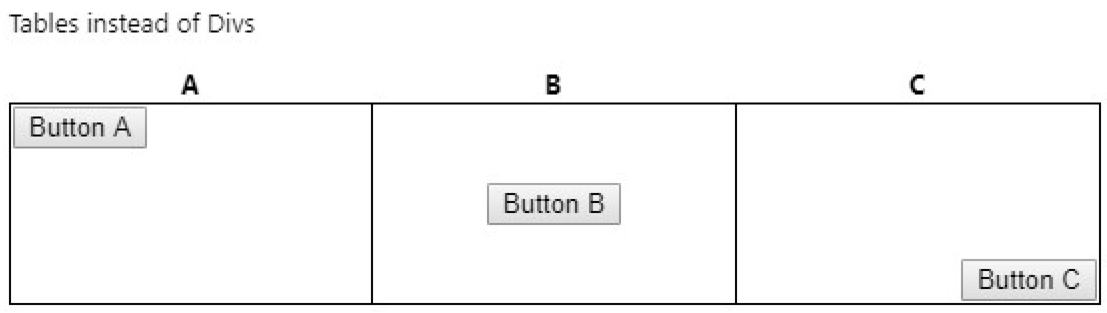
HTML:

```
<table class='w100p noborders'>
```

CSS:

```
.noborders {
  border-collapse: collapse;
}
```

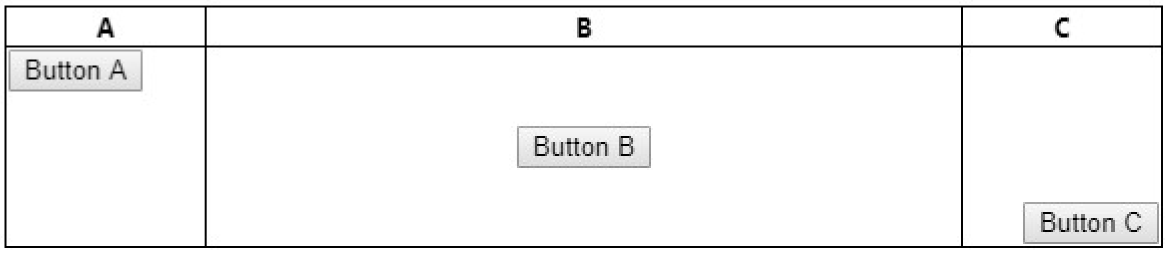
Resulting in:



## Lesson 5 - Hidden Spacing

As with divs, table columns have hidden spacing you may not be aware of.  I'm beginning to realize that it's a good idea to start a UI layout design with borders around *everything* so you can see exactly what's going on in the layout!  This is easily accomplished by specifying the CSS style for your elements, for example:

```
table {
  border-style:solid;
  border-width: 1px;
}

th {
  border-style:solid;
  border-width: 1px;
}

tr {
  border-style:solid;
  border-width: 1px;
}

td {
  border-style:solid;
  border-width: 1px;
}
```



You can then delete this CSS when you're happy with the layout.

## Fixed Width Columns

Here, the `auto` style and the `col` elements come into play.  The following example illustrates the left and right columns having fixed widths and the center column being sized to fit between the two.

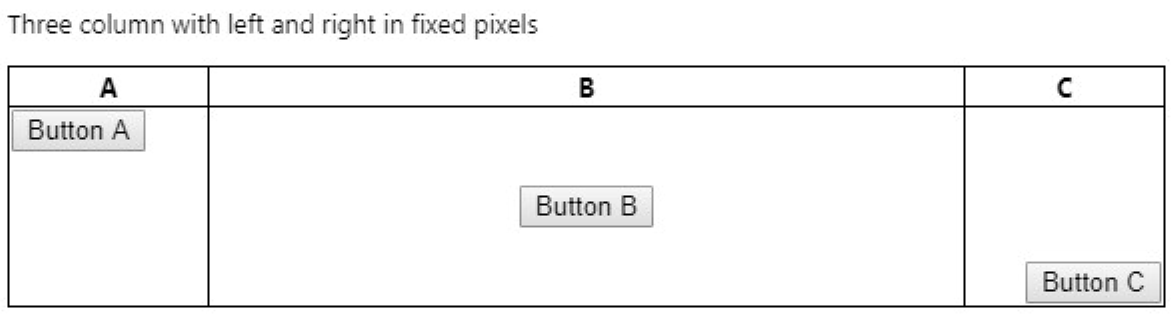
HTML (note I'm using the element CSS described above to set the border for the table, tr, th, and td elements):

```
<p>Three column with left and right in fixed pixels</p>
<table class='w100p noborders' style='table-layout:fixed'>
  <col class='w100px'/>
  <col class='wauto'/>
  <col class='w100px'/>
  <tr>
    <th>A</th>
    <th>B</th>
    <th>C</th>
  </tr>
  <tr class='h100px'>
    <td class='vatop'><button>Button A</button></td>
    <td class='tacenter vacenter'><button>Button B</button></td>
    <td class='taright vabottom'><button>Button C</button></td>
  </tr>
</table>
```

Additional CSS:

```
.w100px {
  width: 100px;
}

.wauto {
  width: auto;
}
```

This resulting in the center column resizing as the browser width is increased / decreased while the left and right columns remain fixed width.


Three column with left and right in fixed pixels

| A | B | C |
|---|---|---|
| Button A | Button B | Button C |

## Column Scrolling

As with divs, let's see if we can get scrolling to work in a column.  The solution is quite simple -- put a `div` inside the `td`! Contrast this with the approach described in the section on divs, particularly note that the height when using divs was specified as 100% while here, it's specified as the column height, 100 *pixels* matching the outer tr height.

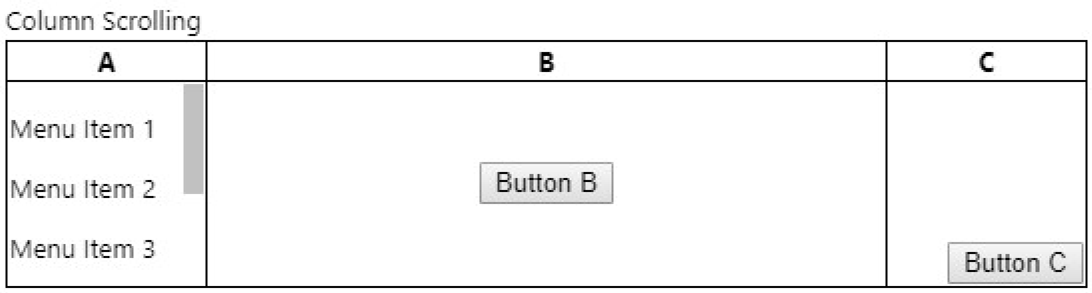
HTML:

```html
<p>Column Scrolling</p>
<table class='w100p noborders' style='table-layout:fixed'>
  <col class='w100px'/>
  <col class='wauto'/>
  <col class='w100px'/>
  <tr>
    <th>A</th>
    <th>B</th>
    <th>C</th>
  </tr>
  <tr class='h100px'>
    <td class='vatop'>
      <div class='scrolly h100px'>
        <p>Menu Item 1</p>
        <p>Menu Item 2</p>
        <p>Menu Item 3</p>
        <p>Menu Item 4</p>
        <p>Menu Item 5</p>
        <p>Menu Item 6</p>
      </div>
    </td>
    <td class='tacenter vacenter'><button>Button B</button></td>
    <td class='taright vabottom'><button>Button C</button></td>
  </tr>
</table>
```

Additional CSS:

```css
.scrolly {
  overflow-y: auto;
}
```

And we get:



## Conclusion

So that's as far I want to take this today. What I've shown here is that you can use divs and tables for layout control equally well, though given that some of div layout requires having the div behave like a table and table cell tends to suggest that using tables is better than divs. Also, some of wonky things like using the `calc()` function are not necessary with tables. I hope you at least had fun reading this and perhaps learned a thing or two. And if there's a better way of doing things, please leave a comment!

## License

## About the Author

**Marc Clifton**

Architect Interacx

United States

Blog: https://marcclifton.wordpress.com/
Home Page: http://www.marcclifton.com
Research: http://www.higherorderprogramming.com/
GitHub: https://github.com/cliftonm

All my life I have been passionate about architecture / software design, as this is the cornerstone to a maintainable and extensible application. As such, I have enjoyed exploring some crazy ideas and discovering that they are not so crazy after all. I also love writing about my ideas and seeing the community response. As a consultant, I've enjoyed working in a wide range of industries such as aerospace, boatyard management, remote sensing, emergency services / data management, and casino operations. I've done a variety of pro-bono work non-profit organizations related to nature conservancy, drug recovery and women's health.

## Comments and Discussions

**6 messages** have been posted for this article Visit **https://www.codeproject.com/Articles/5164453/Fun-Exploring-Div-and-Table-Layout** to post and view comments on this article, or click **here** to get a print view with messages.