A LIST APART

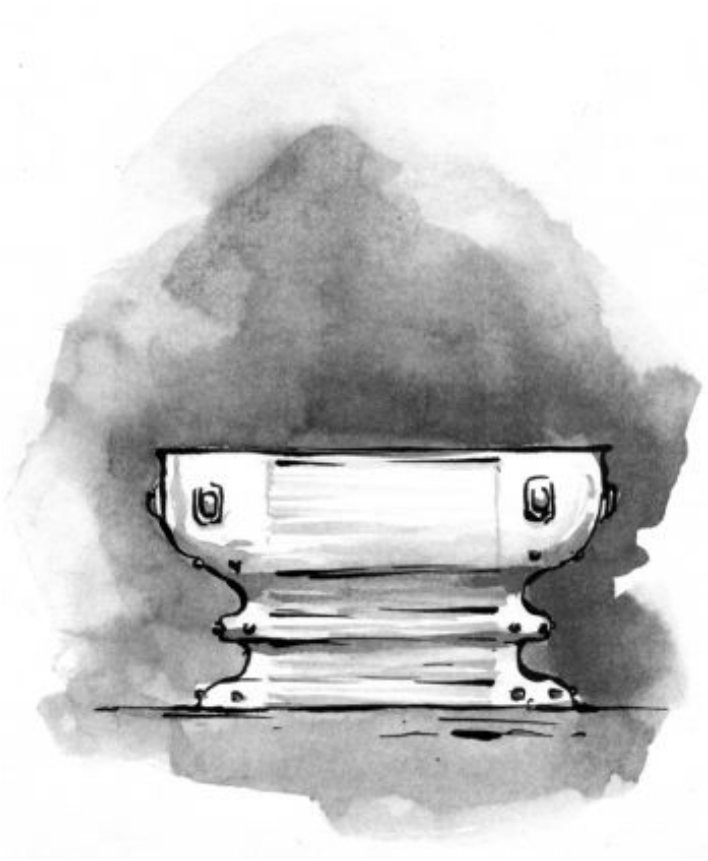


Illustration by **Kevin Cornell**

In Search of the Holy Grail

by **Matthew Levine** · January 30, 2006

Published in CSS, HTML, Layout & Grids, Interaction Design

I'm sorry. Really. I didn't name it. I don't mean to overstate its importance or trivialize the other and rather weightier Holy Grails *(http://en.wikipedia.org/wiki/Holy_Grail)*.

But the name's out there, and we all know what it means.

Three columns. One fixed-width sidebar for your navigation, another for, say, your Google Ads or your Flickr photos—and, as in a fancy truffle, a liquid center for the real substance. Its wide applicability in this golden age of blogging, along with its considerable difficulty, is what has earned the layout the title of Holy Grail.

Many articles have been written about the grail, and several good templates exist. However, all the existing solutions involve sacrifices: proper source order *(http://www.manisheriar.com/holygrail/index.htm)*, full-width footers

(http://glish.com/css/7.asp), and lean markup *(http://www.alistapart.com/articles/negativemargins/)* are often compromised in the pursuit of this elusive layout.

A recent project has brought my personal grail quest to an end. The technique I'll describe will allow you to deploy the Holy Grail layout without compromising your code or your flexibility. It will:

1. have a fluid center with fixed width sidebars,
2. allow the center column to appear first in the source,
3. allow any column to be the tallest,
4. require only a single extra `div` of markup, and
5. require very simple CSS, with minimal ~~hacks~~ patches.

On the shoulders of giants

The technique presented here is inspired by Alex Robinson's brilliant One True Layout *(http://positioniseverything.net/articles/onetruelayout/)*. Alex even addressed the Holy Grail problem in his article, but his solution requires two wrappers and makes padding difficult without a further layer of `div`s within each column.

Another lead came from Eric Meyer's adaptation *(http://meyerweb.com/eric/thoughts/2005/11/09/multi-unit-any-order-columns/)* that uses positioning to mix multiple unit types. His example also yields a three-column layout with fixed sidebars and a liquid center. Unfortunately, it relies on approximate percentages and fills a portion of the viewport that varies widely with different screen resolutions.

Enough talk—let's see some code

The required HTML is intuitive and elegant.

(For the sake of clarity in demonstrating this technique, we are *intentionally* using the non-semantic `id`s "center," "left," and "right." We recommend you use semantic `id`s in any application of this technique. —Ed.)

```
<div id="header"></div><div id="container">
  <div id="center" class="column"></div>
  <div id="left" class="column"></div>
  <div id="right" class="column"></div></div><div id="footer">
</div>
```

That's it. A single extra `div` to contain the columns is all that you need; this satisfies even my obsessive compulsive markup habits.

The stylesheet is almost as simple. Let's say you want to have a left column with a fixed width of 200 pixels and a right column with a fixed width of 150 pixels. To simplify the comments, I'll abbreviate the left, right, and center columns as LC, RC, and CC, respectively. The essential CSS is here:

```
body {
  min-width: 550px;       /* 2x LC width + RC width */
}
#container {
  padding-left: 200px;    /* LC width */
  padding-right: 150px;   /* RC width */
}
#container .column {
  position: relative;
  float: left;
}
#center {
  width: 100%;
}
#left {
  width: 200px;           /* LC width */
  right: 200px;           /* LC width */
  margin-left: -100%;
}
#right {
  width: 150px;           /* RC width */
  margin-right: -150px;   /* RC width */
}
#footer {
  clear: both;
}
/*** IE6 Fix ***/
* html #left {
  left: 150px;            /* RC width */
}
```

Simply replace the values with your desired dimensions and the grail is yours. The technique works in all modern browsers: Safari, Opera, Firefox, and (with the single-rule hack at the bottom) IE6. IE5.5 support would require at least a box-model hack, which is left as an exercise to the reader.

Take a look *(/d/holygrail/example_1.html)* and marvel at the elegance.

How it works

The strategy is straightforward. The container `div` will have a liquid center and fixed-width padding on the side. The trick is then to get the left column to line up with the left padding and the right column with the right padding, leaving the center column to fill the liquid width of the container.

Let's build this up step by step.

STEP 1: CREATE THE FRAME

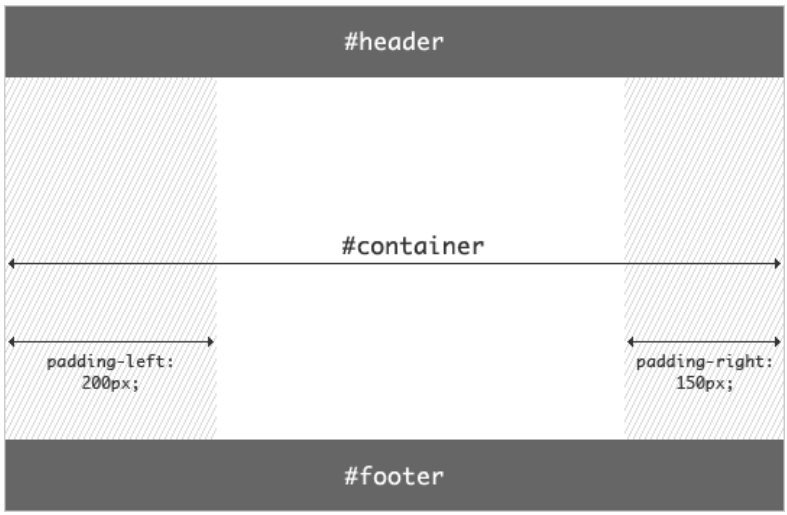
Start with the header, footer, and container.

```
<div id="header"></div><div id="container"></div><div id="footer">
</div>
```

We pad the container with the width we want our left and right columns to occupy.

```
#container {
  padding-left: 200px;   /* LC width */
  padding-right: 150px;  /* RC width */
}
```

Our layout now looks like this:



Step 1: Create the frame

STEP 2: ADD THE COLUMNS

Now that we have our basic frame, we'll stick in the columns.

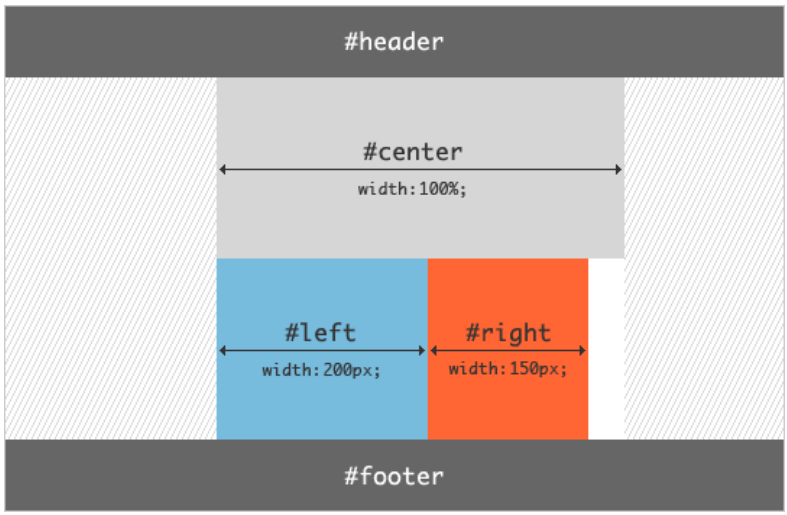
```
<div id="header"></div><div id="container">
  <div id="center" class="column"></div>
  <div id="left" class="column"></div>
  <div id="right" class="column"></div>
</div><div id="footer"></div>
```

Next we add the appropriate widths and float them to get them in line. We'll also need to clear the footer to keep it beneath the floated columns.

```
#container .column {
  float: left;
}
#center {
  width: 100%;
}
#left {
  width: 200px;  /* LC width */
}
#right {
  width: 150px;  /* RC width */
}
#footer {
  clear: both;
}
```

Note that the 100% width on the center column refers to the width of the container div, *exclusive of the padding*. We'll see this 100% width again as the layout comes together, and it will *still* refer to this central width of the container.

The columns now want to line up in order, but because the center column is taking up 100% of the available space, the left and right columns wrap.



Step 2: Add the columns

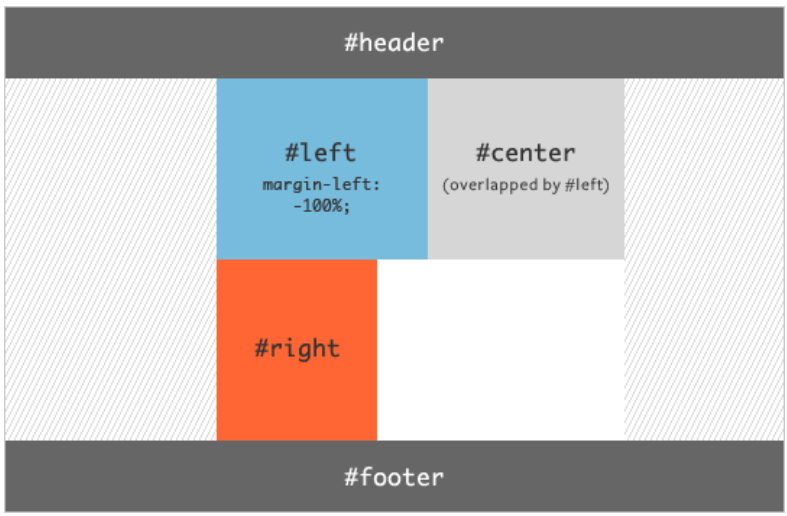
STEP 3: PULL THE LEFT COLUMN INTO PLACE

The only thing left is to get the colums to line up with the padding on the container. The center column starts exactly where it needs to be, so we'll focus on the left column.

It takes two steps to get the left column in place. First, we'll pull it all the way across the center column with a 100% negative margin. Remember that the 100% refers to the central width of the container, which is also exactly the width of the center column.

```
#left {
  width: 200px;         /* LC width */
  margin-left: -100%;
}
```

Now the left column is overlapping the center column, sharing its left edge. The right column floats left and nestles against right edge of the center column (but still wraps), leaving us with the following:

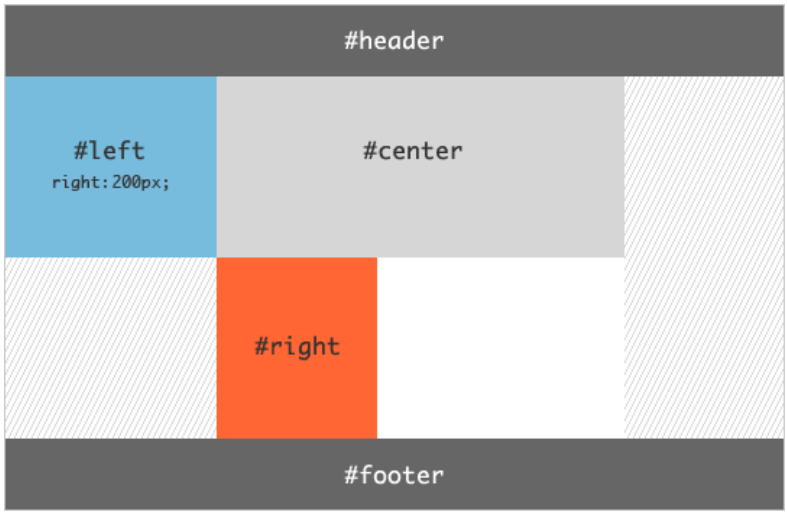


Step 3: Pull the left column into place—halfway there

To push the left column the rest of the way over, we'll use relative positioning with an offset that's exactly the width of the left column.

```
#container .columns {
  float: left;
  position: relative;
}
#left {
  width: 200px;         /* LC width */
  margin-left: -100%;
  right: 200px;         /* LC width */
}
```

The `right` property pushes it 200px *away* from the right edge; that is, to the left. Now the left column lines up perfectly with the left padding of the container.



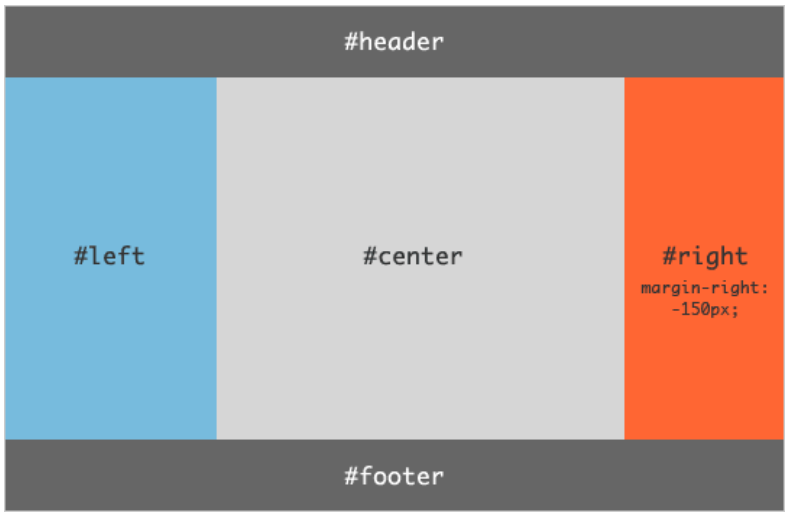
Step 3: Left column pulled into place

STEP 4: PULL THE RIGHT COLUMN INTO PLACE

The only task remaining is to pull the right column into place. To do that, we just need to pull it out of the container and into the container's padding. We'll again use a negative margin.

```
#right {
  width: 150px;           /* RC width */
  margin-right: -150px;   /* RC width */
}
```

Everything is now in its right place, and the wrapping disappears.



Step 4: Pull the right column into place

STEP 5: DESIGN DEFENSIVELY

If the browser window is resized so that the center becomes smaller than the left column, the layout breaks in a standards-compliant browser. Setting a `min-width` on the `body` keeps your columns in place. With IE6 this doesn't happen, so the fact that it doesn't support `min-width` isn't a problem.

```
body {
  min-width: 550px;  /* 2x LC width + RC width */
}
```

Of course, no layout technique would be complete without requiring some sort of workaround in Internet Explorer. The negative margin pulls the left column too far to the left in IE6 (the full width of the browser window). We need to push it back to the right the full width of the right column—using the star-html hack to mask it from other browsers—and we're ready to go.

```
* html #left {
  left: 150px;  /* RC width */
}
```

The reason we need to use the width of the *right* column involves a bit of algebra. I won't bore you with the details; you can work it out for yourself or just consider it another one of IE's many charms.

Padding, please

I'm no designer, but looking at the layout above offends even my aesthetic sensibilities. The unpadded columns are hard on the eyes and difficult to read. We need whitespace.

One of the drawbacks of using percentages with the One True Layout *(http://www.positioniseverything.net/articles/onetruelayout/)* to create liquid columns that it makes padding the columns a bit tricky. Percentage paddings tend to look bad at some screen widths. Fixed paddings can be added, but only by cluttering the markup with a `div` nested inside each column.

With this technique, padding isn't a problem. Padding can be added directly to the left and right columns; just adjust the width accordingly. To give a 10-pixel padding to the left column in the example above, but keep it's full width (padding + width) at 200px, simply change the rule as follows:

```
#left {
  width: 180px;         /* LC fullwidth - padding */
  padding: 0 10px;
  right: 200px;         /* LC fullwidth */
  margin-left: -100%;
}
```

Padding the center requires a little more ingenuity, but no more markup and only a pinch of additional CSS.

The padding plus a width of 100% causes the center column to expand beyond the non-padded width of the container. In order to tame it back into place, we need to increase the right margin by the total amount of the padding. This ensures that the center column is only as large as we expect it to be.

Also, since the center column is now wider, the left column has a larger distance to move in order to get to the correct place. Increasing the offset by the total center padding does the trick.

To make this concrete, I'll modify the example to add a 10-pixel padding to each side column (for a total of 20 pixels), and a 20-pixel padding to each side of the center (for a total of 40 pixels). The new CSS looks like this:

```
body {
  min-width: 630px;       /* 2x (LC fullwidth +
                             CC padding) + RC fullwidth */
}
#container {
  padding-left: 200px;    /* LC fullwidth */
  padding-right: 190px;   /* RC fullwidth + CC padding */
}
#container .column {
  position: relative;
```

```
  float: left;
}
#center {
  padding: 10px 20px;    /* CC padding */
  width: 100%;
}
#left {
  width: 180px;          /* LC width */
  padding: 0 10px;       /* LC padding */
  right: 240px;          /* LC fullwidth + CC padding */
  margin-left: -100%;
}
#right {
  width: 130px;          /* RC width */
  padding: 0 10px;       /* RC padding */
  margin-right: -190px;  /* RC fullwidth + CC padding */
}
#footer {
  clear: both;
}/*** IE Fix ***/
* html #left {
  left: 150px;           /* RC fullwidth */
}
```

Of course, top and bottom padding can be added without any problems. See this nicely padded version *(/d/holygrail/example_2.html)* for the full template.

This technique works just as well for ems. Unfortunately, you can't mix-and-match ems and pixels, so choose, but choose wisely.

Equal-height columns

This technique really comes together when the columns are given equal heights. The method I'm using is adapted wholesale *(http://www.positioniseverything.net/articles/onetruelayout/equalheight)* from the One True Layout, so I won't go over it in detail. To deploy it, simply add the following CSS:

```
#container {
  overflow: hidden;
}
#container .column {
  padding-bottom: 20010px;  /* X + padding-bottom */
  margin-bottom: -20000px;  /* X */
```

```
}
#footer {
  position: relative;
}
```

Here, I've given the columns an extra padding of 10px on the bottom.

The usual caveats apply. Be aware that Opera 8 has a bug with `overflow: hidden` that leaves all of your columns huge. A workaround is detailed on the One True Layout page; you can use that, or wait for Opera 9 (which fixes the bug) to come out of beta.

One additional problem unique to this layout is that IE doesn't clip the column backgrounds at the bottom of the container. They spill out over the footer if the page is not as tall as the viewport. This isn't a problem if you don't have a separate footer, or if your pages are tall enough to ensure that you'll always take up the whole viewport.

If you need that footer, though, never fear. This, too, is fixable, but requires one more `div`. Add a wrapper to the footer, like so:

```
<div id="footer-wrapper">
  <div id="footer"></div>
</div>
```

Now re-use the same trick with the equal columns to make the footer wrapper extend beyond the page, leaving the footer itself for you to do with as you please.

```
* html body {
  overflow: hidden;
}
* html #footer-wrapper {
  float: left;
  position: relative;
  width: 100%;
  padding-bottom: 10010px;
  margin-bottom: -10000px;
  background: #fff;          /* Same as body
                                background */
}
```

This solves the problem, and leaves us with the desired result *(/d/holygrail/example_3.html)* and a minimum of cruft.

Oh, and there's one more thing

The extremists out there might be wondering to themselves whether there's not an even *better* way to do this. After all, the method I've illustrated introduces a non-semantic container `div`. Surely we can't have *one extra* `div` cluttering up our otherwise flawless markup.

If you, like me, have been wondering this very thing, wonder no more. As a special bonus, I present to you the wrapper-free Holy Grail *(/d/holygrail/example_4.html)*, in all its minimalist glory. One `div` for each section of the code—no more, no less. Semantic bliss, almost worthy of the title of "Holy Grail."

The principle behind the CSS is the same. The padding has been applied directly to the body, eliminating the need for any containers. Negative margins stretch the header and footer to ensure that they take up the entire space.

This tiny version works in all the aforementioned browsers, even (almost shockingly) Internet Explorer. Equal-height columns do *not* work, however, and this layout will break down for very small window widths. Use it with caution.

So what now?

While this particular application of the Holy Grail is rather specific, the technique can be generalized considerably. Why not have two liquid columns? Why not switch up the column orders? These applications are beyond the scope of this article, but achievable with minor modifications. Use the grail wisely, and it can be a particularly handy (and clutter-free) addition to your bag of CSS tricks.

About the Author

Matthew Levine

Before working as a full-time web developer, Matthew taught elementary school *(http://www.teachforamerica.org)* and studied physics and philosophy. When he isn't streamlining websites *(http://www.infocraft.com/)* he spends his time in the bayous of South Louisiana perfecting his cajun two-step.