PAGE LAYOUT WITH FLOATS AND POSITIONING

For use with:
Learning Web Design, 5e

by Jennifer Robbins

CSS Grid and Flexbox (see **Note**) were designed from the ground up to give designers control over page layout, and now that those standards have solid browser support, they are definitely the way to go moving forward. However, we are still in a time of transition. Older, non-supporting browsers (Internet Explorer in particular) have a way of stubbornly hanging around, and as long as they continue to show up in significant numbers in our visitor statistics, we need to provide reasonable fallbacks for our Grid- and Flexbox-based designs. That's where knowing the old float- and position-based layout techniques may still come in handy.

I've made this article (originally a chapter from the fourth edition of *Learning Web Design*) available should you need to support old browsers that support neither Grid nor Flexbox. Otherwise, these techniques should be considered obsolete and should be avoided in favor of proper layout standards.

This article contains templates and techniques for the following:

- Two- and three-column layouts using floats
- A source-independent layout using floats and negative margins
- A multicolumn layout using positioning

To get the most out of the examples, you will need a solid understanding of how floating and positioning work, as presented in **Chapter 15, Floating and Positioning**, in *Learning Web Design, 5e*.

The examples in this article are intended to be a "starter kit." They should give you a good head start toward understanding how layout works, but they are not universal solutions. The templates presented here are simplified and may not work for every situation, although I've tried to point out the relevant shortcomings of each. Your content may dictate more complicated solutions.

IN THIS ARTICLE

NOTE

I cover Grid and Flexbox in detail in **Chapter 16, CSS Layout with Flexbox and Grid**, *of Learning Web Design, 5e (O'Reilly).*

MULTIPLE COLUMNS USING FLOATS

The truth is, floats were never intended to be a page layout tool. But just as tables were co-opted before them, the development community put floats to use for columns because it was one of the only options we had. These days, we have better options, but if you need to support older browsers, you may still want to have a few floated column tricks up your sleeve.

The advantages that floats have over absolute positioning for layout are that they prevent content from overlapping other content, and they make it easier to keep footer content at the bottom of the page. The drawback is that they are dependent on the order in which the elements appear in the source, although there is a workaround using negative margins, as we'll see later.

This section provides templates and techniques for creating a variety of standard column page layouts using floats, including the following:

- A two-column fluid layout
- A two-column layout with a fixed width
- A two-column layout with a fixed width, centered
- A three-column fluid layout
- A three-column layout that is not tied to the source order of the document

How to Use the Examples

The sample pages in this section aren't pretty. In fact, I've stripped them down to their bare minimum to help make the structure and strategy as clear as possible. Here are a few notes regarding the templates and how to use them.

Simplified markup and styles

I've included only the bare minimum markup and styles in the examples—just enough to follow how each layout is created. All style rules not related to layout have been omitted to save space and to keep the focus on what is needed to move elements around.

Headers and footers

I've included a header and footer on many of these examples, but either one or both could easily be omitted for a minimal two- or three-column layout.

Color-coding

I've added outlines around each column (using the **`outline`** property) so you can see the edges of the floated or positioned elements in the layout. The outlines are color-coded with the markup and the styles that create them in an effort to make the connections clearer.

Padding and borders

To keep width calculations simple, I've avoided using padding and borders on the column elements in the examples. Setting the box-sizing model to **`border-box`** will allow you to use padding and margins without recalculating widths. However, if you use the **`content-box`** model in your designs, be sure to compensate for the extra width so the overall width does not exceed 100%.

Customization

There is obviously a lot more that could be done with text, backgrounds, margins, padding, and borders to make these pages more appealing. Once you've laid a framework with these templates, you should feel free to change the measurements and add your own styles.

Two Columns, Fluid Layout

In this section, we'll look at two methods for creating a fluid layout. In fluid layouts, the page area and columns resize proportionally to fill the available space in the browser, consistent with the behavior of the normal flow. Widths in fluid layouts are generally specified in percentage values; however, it is also possible to make one or more columns a fixed width and allow the other column(s) to fill the remaining width of the viewport (also known as a hybrid layout).

I've included two techniques for creating two-column, fluid layouts:

- Floating one element and using a margin on the second to clear a space for it.
- Floating all the elements to one side. You can float columns to the left or right depending on the source order in your document and where you want each column to appear on the page.

One float with a margin

Let's start with a simple one-float approach.

THE STRATEGY

Float one column element to the right or left and add a margin on the remaining column element. Clear the footer element to keep it at the bottom of the page. The underlying structure and resulting layout is shown in FIGURE A.

CSS Outlines

In the examples in this section, I've taken advantage of the **`outline`** property to reveal the edges of the floated and positioned columns. Outlines look like borders, and the syntax is the same, but there is an important difference. Outlines, unlike borders, are not calculated in the width of the element box. They just lay on top, not interfering with anything. This makes outlines a great tool for checking your layout work because you can turn them on and off without affecting your width measurements.

The outline declaration looks a lot like the declaration for borders:

```
div#links {
  outline: 3px dashed red;
}
```

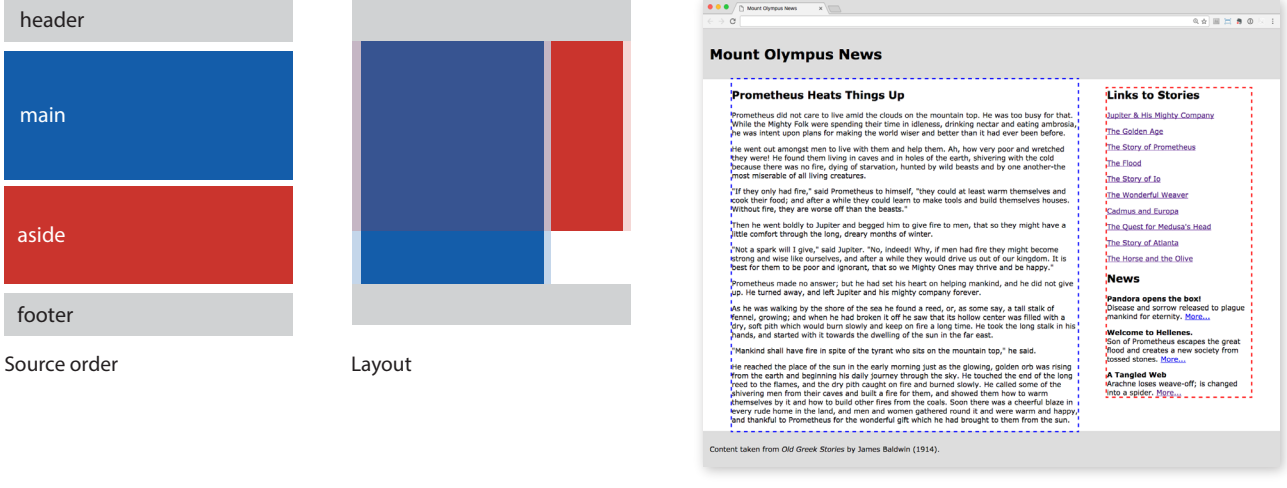


FIGURE A. Two columns created with one float and a margin.

THE MARKUP

```
<header>Masthead and headline</header>

<main>Main article</main>

<aside>List of links and news</aside>

<footer>Copyright information</footer>
```

THE STYLES

```
main {
  float: left;
  width: 60%;
  margin: 0 5%;
}
aside {
  width: 25%;
  margin-left: 70%;
}
footer {
  clear: left;
}
```

NOTES

This one is pretty straightforward, but because this is our first one, I'll point a few things out:

- Remember that I've omitted the styles for the header, footer, and text to keep the examples as simple as possible. Keep in mind that there is a bit more at work in here than what is listed under **"The styles"** (nothing you couldn't figure out, though: background colors, padding, stuff like that). There are also 3px dashed outlines applied to the `main` and `aside` elements to make the structure clearer in the figure. This is true for all examples in this article.
- The source document has been divided into elements: `header`, `main`, `aside`, and `footer`. The markup shows the order in which they appear in the source.
- The `main` element has been floated to the left and given a width of 60% with 5% margins on the left and right sides.
- The `aside` element has been wrapped around the `main` element. It has a left margin of 70%, enough to make room for the main element box (60% box width plus 10% margin width).
- The `footer` is cleared so it starts below the floated content.

Floating both columns

THE STRATEGY

Set widths on both column elements and float them to the left. Clear the footer to keep it at the bottom of the page. The underlying structure and resulting layout is shown in FIGURE B.

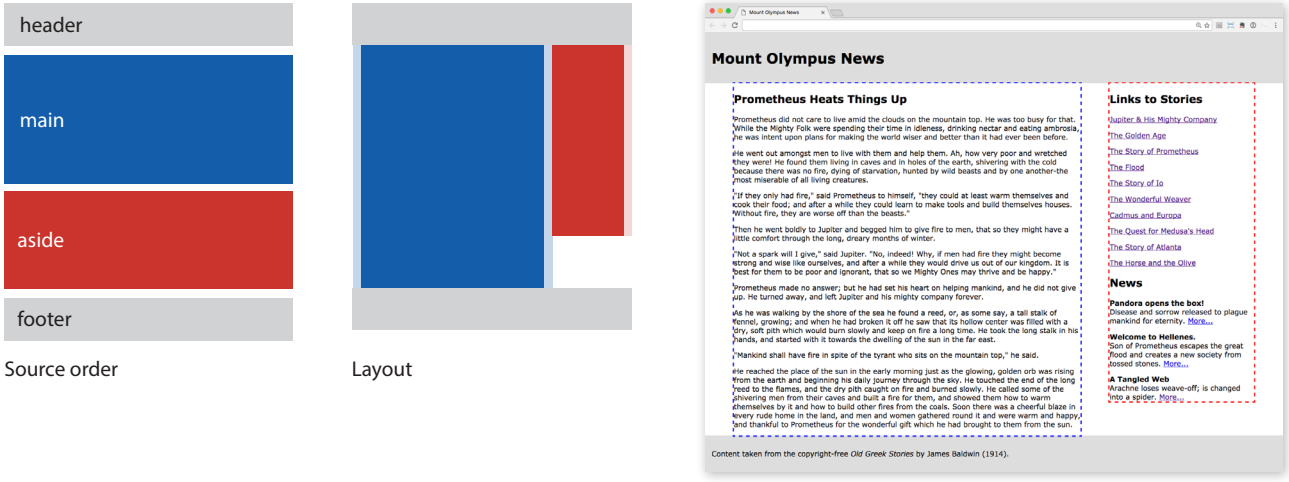


FIGURE B. Floating two columns.

THE MARKUP

```
<header>Masthead and headline</header>
<main>Main article</main>
<aside>List of links and news</aside>
<footer>Copyright information</footer>
```

THE STYLES

```
main {
  float: left;
  width: 60%;
  margin: 0 5%;
}
aside {
  float: left;
  width: 25%;
  margin: 0 5% 0 0;
}
footer {
  clear: left;
}
```

NOTES

This one has the same visual result as the first two-column example, which goes to show you that there are often multiple approaches to a single design goal. In this approach:

- Both **main** and **aside** have been floated to the left. Because they are floats, widths were specified for each. You can make your columns as wide as you like.
- The **main** element has a 5% margin applied on the left and right sides. The **aside** element needs a margin only on the right. The margins on the top have been set to 0 so they vertically align.
- The **footer** is cleared so it starts below the floated content.

> **TIP**
>
> You could also float one column to the left and the other to the right for the same effect.

Two Columns, Fixed Width

This time, we'll make the layout fixed width instead of fluid. Fixed-width layouts, as the name implies, are set at a specific pixel width. They are less useful on smaller mobile devices where fluid layouts provide more flexibility; but on larger, desktop monitor sizes, they can prevent content from getting too wide to read comfortably.

I'll show you two options:

- A left-aligned fixed-width layout (default)
- A centered fixed-width layout

Left-aligned, fixed-width layout

THE STRATEGY

Wrap the entire page in a **div** set to a specific pixel width. We'll specify pixel values for the floated elements as well, but the floating and clearing method is the same. The resulting layout is shown in FIGURE C.

THE MARKUP

```
<div id="wrapper">
  <header>Masthead and headline</header>
  <main>Main article</main>
  <aside>List of links and news</aside>
  <footer>Copyright information</footer>
</div>
```

THE STYLES

```
#wrapper {
  width: 960px;
}
main {
  float: left;
  width: 650px;
  margin: 0 20px;
}
aside {
  float: left;
  width: 250px;
  margin: 0 20px 0 0;
}
footer {
  clear: left;
}
```

NOTES

- All of the content is contained in a **div** (**#wrapper**) that has been set to 960 pixels wide.

Say "Enough Is Enough" with max-width

Fluid layouts are great because they can adapt themselves to the screen or browser window size on which they are displayed. We spend a lot of time considering how our pages fare in small spaces, but don't forget that at the other end of the spectrum are high-resolution monitors approaching or exceeding 2,000 pixels in width. Users may not maximize their browser windows to fill the whole screen, but there is the potential for the browser window to be so wide that the text in your flexible columns becomes difficult to read.

You can put a stop to the madness with the **max-width** property. Apply it to the column element you are most concerned about becoming unreadable (like the **main** column in the **"Two Columns, Fluid Layout"** example), or put the whole page in a wrapper element and put the brakes on the width of the whole page.

Similarly, the **min-width** property is available if you want to prevent your page from looking too scrunched.

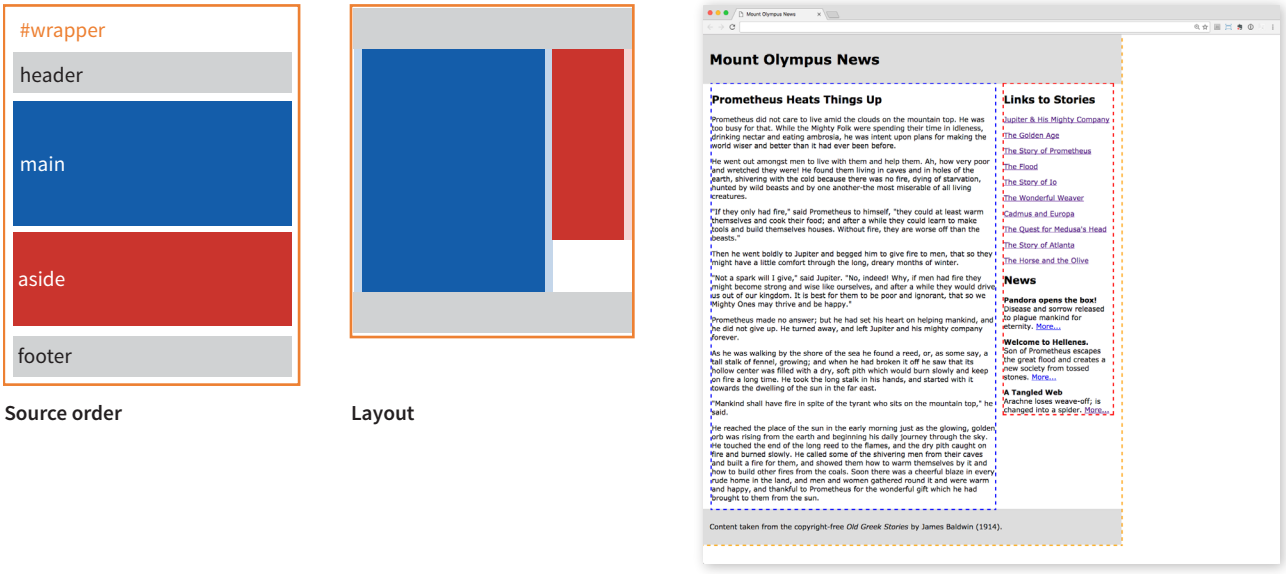


FIGURE C. A fixed width, two-column layout using floats.

- I've changed the widths and margins to pixel measurements as well, taking care not to exceed a total of 960. If they added up to more than the width of the `#wrapper` container, we'd get the dreaded float drop. Keep in mind that if you add padding or borders, the total of their widths would need to be subtracted from the width values to keep the total width the same unless you use the `border-box` box-sizing model.

Centered, fixed-width layout

At this point, it's really easy to center the fixed-width layout.

THE STRATEGY

Set the left and right margins on the `#wrapper` container to `auto`, which keeps the whole page centered. The markup is exactly the same as in the previous example. We only need to add a `margin` declaration to the styles. Easy as pie. The resulting layout is shown in FIGURE D.

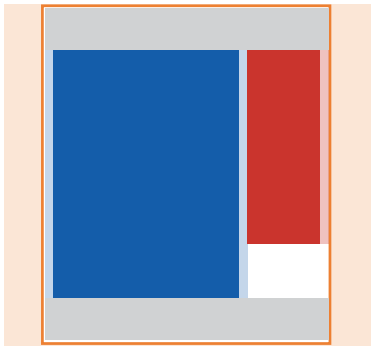
THE STYLES

```
#wrapper {
  width: 960px;
  margin: 0 auto;
}
```

NOTES

- The `auto` margin setting on the left and right sides keeps the `#wrapper` centered in the browser window.

Source order



Layout

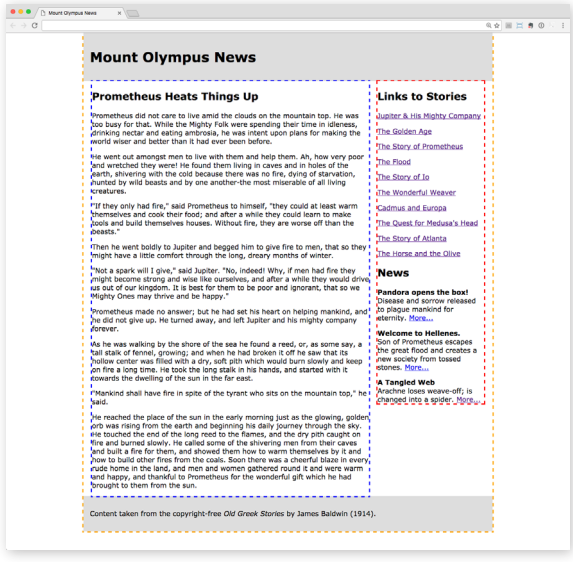


FIGURE D. Our fixed-width layout is now centered in the browser window.

Full-Width Headers and Footers

If you want the **header** and **footer** to be the full browser width, but also want to keep the content between them fixed width and centered (FIGURE E), change the markup so that only the **main** and **aside** elements are inside the **#wrapper**.

```
<header>Masthead and headline</header>
<div id="wrapper">
  <main>Main article</main>
  <aside>List of links and news</aside>
</div>
<footer>Copyright information</footer>
```

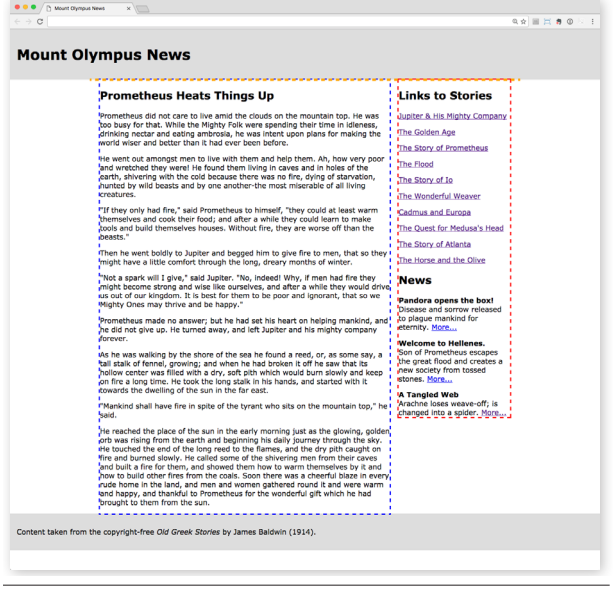


FIGURE E. The header and footer fill the width of the browser, but the content between them remains a fixed width.

Three Columns, Fluid Layout

I suspect you're getting the hang of it so far. Now we'll tackle three-column layouts, which use the same principles but take a little extra finagling. In this example, we'll float all of the elements to the left. You will see that we are tied to the order in which the three floated elements appear in the source.

THE STRATEGY

Set widths on all three-column elements and float them to the left. Clear the footer to keep it at the bottom of the page. The underlying structure and resulting layout is shown in FIGURE F.

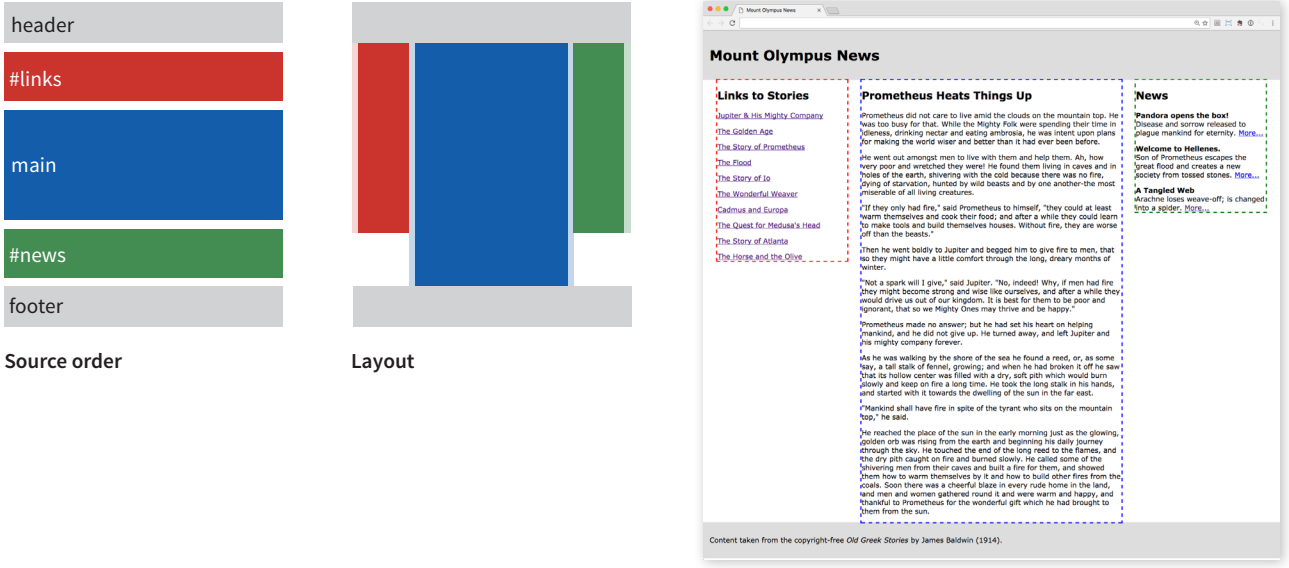


FIGURE F. A fluid-width, three-column layout using three floats.

THE MARKUP

```
<header>Masthead and headline</header>
<div id="links">List of links</div>
<main>Main article</main>
<div id="news">News items</div>
<footer>Copyright information</footer>
```

NOTES

- The markup shows that we now have a total of five sections in the document: **header**, **#links**, **main**, **#news**, and **footer**.
- Using floats alone, if we want the main content column to appear in the middle between the links and news columns, then the **main** element needs to appear between the **#links** and **#news** **div**s in the source. All three columns are given widths and floated to the left. Care must be taken to ensure that the total of the **width** and **margin** measurements is not greater than 100%.

THE STYLES

```
#links {
  float: left;
  width: 22.5%;
  margin: 0 0 0 2.5%;
}
main {
  float: left;
  width: 45%;
  margin: 0 2.5%;
}
#news {
  float: left;
  width: 22.5%;
  margin: 0 2.5% 0 0;
}
footer {
  clear: left;
}
```

■ TIP

Floating the Other Direction

You could also arrange the columns so "News" is on the left and "Links" is on the right by floating the three elements to the right instead of the left. FIGURE G shows the results of floating the column elements to the right inside a fixed-width, centered layout.

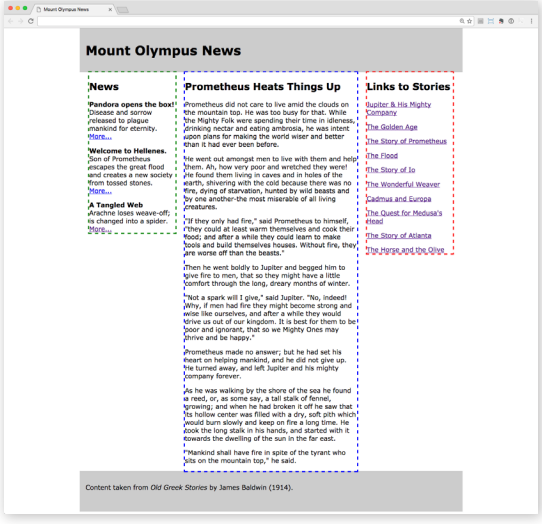


FIGURE G. The resulting fixed-width layout with swapped side columns.

"Any Order" Columns Using Negative Margins

When float-based layouts were beginning to gain steam, many designers wondered, "Is there a way to do three-column floats that is independent of the source order?" It turns out the answer was "Yes!" The trick is to use the magic of negative margin values and a heaping tablespoon of math (a little bit of math never hurt anyone, right?). The technique was first brought to light by Alex Robinson in his classic 2005 article "In Search of the One True Layout" (*positioniseverything.net/articles/onetruelayout/*). See also Matthew Levine's article "In Search of the Holy Grail" on A List Apart (*alistapart.com/article/holygrail*).

■ LAYOUT TIP

You can avoid this whole ordering conundrum by using Flexbox or Grid for columns instead of hacking floats, as this "holy grail" approach clearly does. You can change the display order of flex items and to place grid items in any cell area you choose.

THE STRATEGY

Apply widths and floats to all three column elements, and use a negative margin to "drag" the third column across the page into the left position. The underlying structure and resulting layout is shown in FIGURE H. Notice that although **`main`** comes first in the source, it is in the second-column position. In addition, the **`#links div`** (last in the source) is in the first-column position on the left. This example is fixed, but you can do the same thing with a fluid layout by using percentage values.

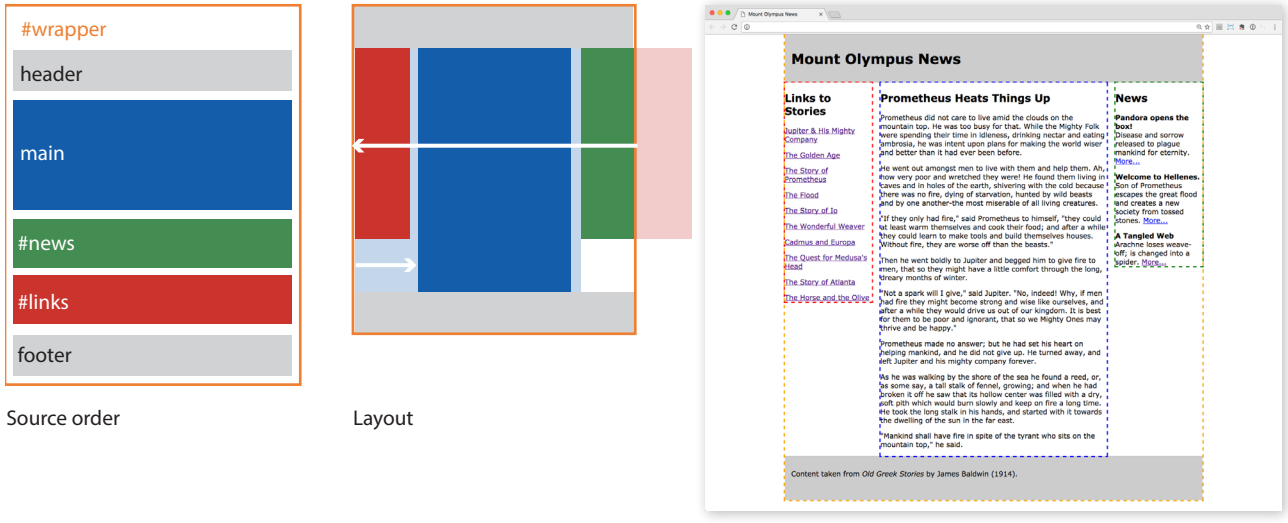


FIGURE H. A fixed-width, three-column layout using three floats. It looks like the previous example, but it is special in that the column order is not the same as the source order.

THE MARKUP

```
<div id="wrapper">
  <header>Masthead and headline</header>
  <main">Main article</main>
  <div id="news">News items</div>
  <div id="links">List of links</div>
  <footer>Copyright information</footer>
</div>
```

THE STYLES

```
#wrapper {
  width: 960px;
  margin: 0 auto;
}
main {
  float: left;
  width: 520px;
  margin-top: 0;
  margin-left: 220px;
  margin-right; 20px;
}
#news {
  float: left;
  width: 200px;
  margin: 0;
}
#links {
  float: left;
  width: 200px;
  margin-top: 0;
  margin-left: -960px;
}
footer {
  clear: left;
}
```

NOTES

This one requires a bit more explanation, so we'll look at how it's done one step at a time.

In the markup, we see that **`main`** comes first, presumably because it is the most important content, and **`#links`** comes last. The whole page is wrapped in a **`div`** (**`#wrapper`**) so that it can be set to a specific width (960px). In the layout, however, the order of the columns from left to right is **`#links`** (200px wide), **`main`** (520px wide), then **`#news`** (200px wide). This layout has 20 pixels of space between columns.

The first step to getting there is moving the **`main`** section to the middle position by applying a left margin that pushes it over enough to make room for the left column (200px) plus the space between (20px); so, **`margin-left: 220px`**. While we're at it, we'll add a 20px right margin on **`main`** as well to make room on its right side. FIGURE 1 shows how the page looks after we apply styles to **`main`**.

Next—and this is the cool part—pull the content that you want to go in the left column (**`#links`**, in this case) to the left by using a *negative* margin value. The trick is figuring out how far to the left it needs to be moved. If you look at FIGURE 1, you can see a ghostly version of **`#links`** that shows where it *wants* to be if the **`#wrapper`** were wide enough. I find it useful to look at the layout in that way because it makes it clear that we need to pull **`#links`** to the left by the widths of all the element boxes ahead of it in the source.

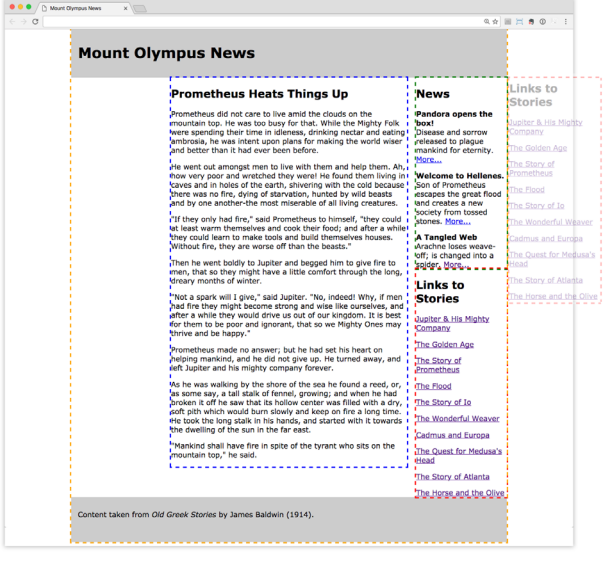


FIGURE 1. The layout after margins are applied to the middle (**`main`**) column element. The shaded box on the right shows where **`#links`** would like to be if it weren't forced under **`#news`**.

In this example, the element box width for **main** is 520px + 220px for the left margin + 20px for the right margin, for a total of 760 pixels. The total width of **#news** is 200px (no margins are applied). That means that the **#links div** needs to be pulled a total of 960 pixels to the left to land in the left-column slot (**margin-left: -960px;**). When the negative margin is applied, **#links** slides into place, and we have the final layout shown in FIGURE H.

If we wanted the **#news div** to be in the left column (FIGURE J), we'd take the same approach: float all the elements to the left, give the **#main** element a wide left margin (220px), and then use a negative margin to pull **#news** to the left. The left margin on **#news** needs to be **-760px** (220 + 520 + 20) to move it across the width of the **#main** element and its two side margins.

WARNING

When you are doing this on your own, remember to include padding and borders into the total element box width calculations as well, unless you are using the **border-box** *box-sizing model.*

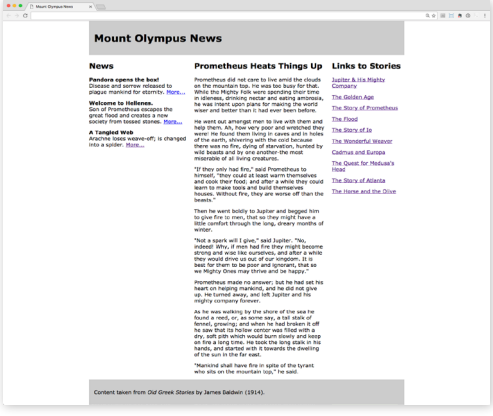


FIGURE J. Floating the News **div** to the left column with a margin of –760px.

POSITIONED LAYOUT

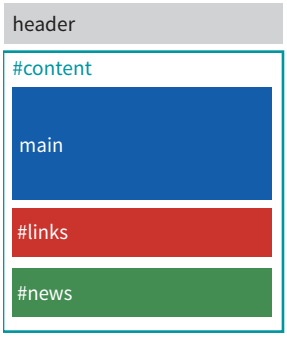
I think we've got floated columns covered. The other way to create columns in a layout is to use absolute positioning. In this section, we'll use positioning to arrange three columns in both fluid and fixed-width pages.

Note that in both examples, I have omitted the **footer** element. I've done this for a couple of reasons. First, when you position all of the elements in a layout, as we will in these examples, they no longer "participate in the layout," which means there is nothing to hold a footer at the bottom of the page. It rises right up to the top. There are solutions to this problem using JavaScript, but they are beyond the scope of this article.

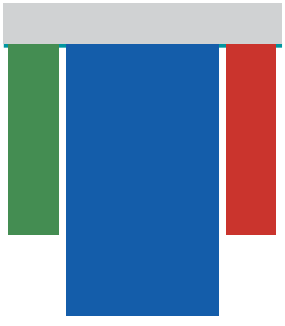
But say we position only the two side columns and let the main center column stay in the flow to hold the footer down. This is certainly a possibility, but if either of the side columns grows longer than the center column, it will overlap the footer content. Between leaping footers and potential overlaps, it's just kind of messy, which is why I've chosen to omit the footer here (and why floats are the more popular layout technique).

Three Columns, Positioned, Fluid Layout

This layout uses percentage values to create three flexible columns. The resulting layout is shown in FIGURE K.



Source order



Layout

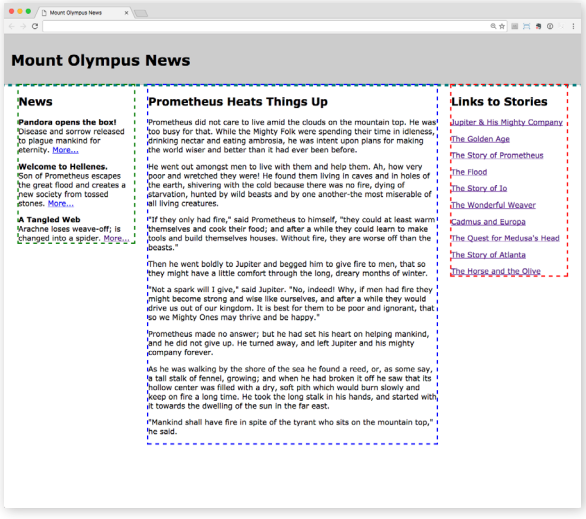


FIGURE K. Three positioned, fluid columns.

THE STRATEGY

Wrap the three sections (**main**, **#news**, **#links**) in a **div** (**#content**) to serve as a containing block for the three positioned columns. Then give the column elements widths and position them in the containing **#content** element.

THE MARKUP

```
<header>Masthead and headline</header>
<div id="content">
  <main>Main article</main>
  <div id="news">News items</div>
  <div id="links">List of links</div>
</div>
```

THE STYLES

```
#content {
  position: relative;
  margin: 0;
}
```

```
main {
  width: 50%;
  position: absolute;
  top: 0;
  left: 25%;
  margin: 0;
}
#news {
  width: 20%;
  position: absolute;
  top: 0;
  left: 2.5%;
  margin: 0;
}
#links {
  width: 20%;
  position: absolute;
  top: 0;
  right: 2.5%;
  margin: 0;
}
```

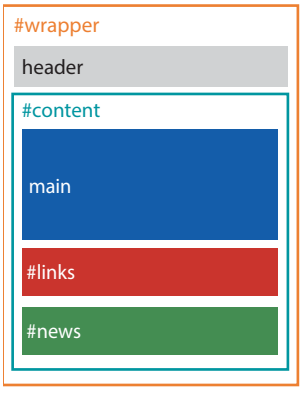
NOTES

I think that you'll find the styles for this layout to be fairly straightforward.

- I created the **#content** containing block to position the columns because we want the columns to always start below the **header**. If we positioned them relative to the browser window (the initial containing block), they may be in the wrong spot if the height of the header should change, such as the result of the **h1** text changing size. Make the **#content div** a containing block by applying the declaration **position: relative**.
- The **main** element is given a width of 50%, and absolute positioning is used to place it at the top of the **#content div** and 25% from the left edge. This will accommodate the 20% width of the left column plus the 2.5% space that serves as a margin to the left and right of it.
- The **#news div** is positioned at the top of the **#content div** and 2.5% from the left edge (**top: 0; left: 2.5%;**).
- The **#links div** is positioned at the top of the **#content div** and 2.5% from the right edge (**top: 0; right: 2.5%;**). No need to calculate the position from the left edge...just put it on the right! Note that we could have positioned the **#news** and **#links** columns flush against their respective edges and used padding to make a little space on the sides. There are usually multiple ways to approach layout goals.
- The only trick to getting this right is making sure your **width** and **margin** measurements do not exceed 100%. You may need to factor in padding and borders as well.

Three Columns, Positioned, Fixed

If you have a use case that requires pixel-level control over your positioned layout, that's pretty easy to do, as we'll see in this example (FIGURE L). It differs from the previous fluid example in that the whole page is contained in a **#wrapper** so it can be fixed and centered, and pixel values are used for the measurements. To save space, I'll just show you the resulting styles here. The markup and positioning strategy are the same.



Source order

Layout

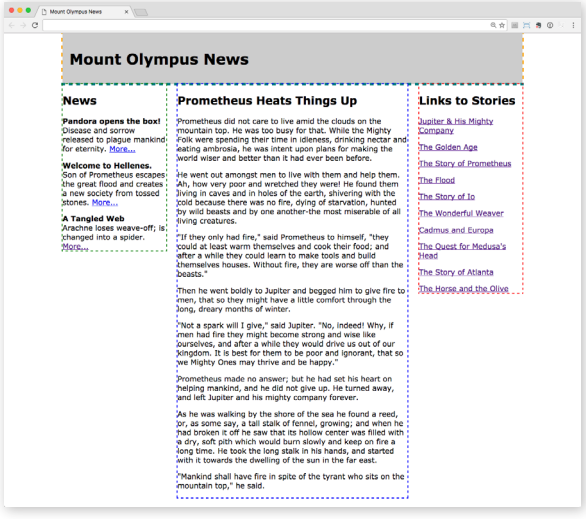


FIGURE L. Three positioned columns in a centered, fixed-width page.

THE STYLES

```
#wrapper {
  width: 960px;
  margin: 0 auto;
}
#content {
  margin: 0;
  position: relative;
}
main {
  width: 520px;
  position: absolute;
  top: 0;
  left: 220px;
  margin: 0;
}
```

```
#news {
  width: 200px;
  position: absolute;
  top: 0;
  left: 0;
  margin: 0;
}
#links {
  width: 200px;
  position: absolute;
  top: 0;
  right: 0;
  margin: 0;
}
```

TOP-TO-BOTTOM COLUMN BACKGROUNDS

Adding color to columns is an effective way to emphasize the division of information and bring a little color to the page. But if you look at the dashed borders in all the screenshot examples we've seen so far, you'll see that column elements often stop well before the bottom of the page. This is one disadvantage of using floats and positioning for columns. If you want to apply backgrounds from top to bottom in columns, you need to get tricky and use tiling background images in a technique known as "faux columns."

NOTE

The problem of uneven columns is solved in CSS Grid and Flexbox, where you can stretch items to fill the available height.

The Faux-Columns Hack

Where there's a will, there's a way, and that has certainly been true in web development. Undeterred by the fact that there was no way to make floated columns fill the height of the page, developers devised a hack to make them *look* like they do. The faux-columns technique uses a wide, thin image with bands of color that correspond to each column. The image tiles vertically in the background of the container that holds the layout, and the columns of content appear in the foreground.

The faux-columns trick is easier to apply to fixed-width layouts, so we'll start there (we'll get to fluid column backgrounds in a moment). FIGURE M shows an image, *two_columns.png*, applied to a fixed-width, two-column layout. I've made *two_columns.png* thick enough that you can see it in the figure, but you could create the image at only 1 pixel high to keep its file size at a minimum and you'd get the same effect.

```
#wrapper {
  width: 960px;
  margin: 0 auto;
  background-image: url(two_columns.png);
  background-repeat: repeat-y;
}
```

You may recognize the layout in FIGURE M as the two-column, fixed-centered layout we made earlier in FIGURE D. This time, it has the *two_columns.png* graphic tiling vertically in the **#wrapper** element.

■ TIP

If your layout lacks a **footer** element to hold the container element open after the columns are floated, apply **overflow: hidden;** to the **#wrapper** to make it stretch around the floats.

two_columns.png

FIGURE M. A tiling background image creates a colored column effect.

Faux Columns for Fluid Layouts

Now that you understand the basic technique, you may be wondering how to make it work for columns with flexible widths. The secret is a *really, really* wide background graphic and the **background-position** property.

We may not know the exact width of the columns in a fluid layout, but we *do* know the point at which the columns are divided. Let's use the two-column fluid example from FIGURE B as an example. The column division occurs 67.5% from the left edge (5% left margin + 60% **main** column width + 2.5%, which is half of the 5% space between margins).

In your image editor of choice, create a horizontal image that is wider than any monitor is likely to go—5,000 pixels ought to do it. Because the graphic needs to be only a few pixels high and is likely to be made up of a few flat colors, the file size should stay pretty small. When you create the column colors, make sure they match the proportion of your columns. In our example, the left-column background should fill 67.5% of the width of the graphic (67.5% × 5,000 = 3,375 pixels).

Apply the wide image as a background pattern to the **body** element, and use **background-position** to align the point where the color changes in the graphic (67.5%) with the point where the columns divide on the page (also 67.5%). In that way, the column break in the image will always be centered in the space between the columns (FIGURE N). And there you have it—faux columns that expand and contract with the column widths.

```
body {
  background-image: url(two_cols_5000px.png);
  background-repeat: repeat-y;
  background-position: 67.5%;
}
```

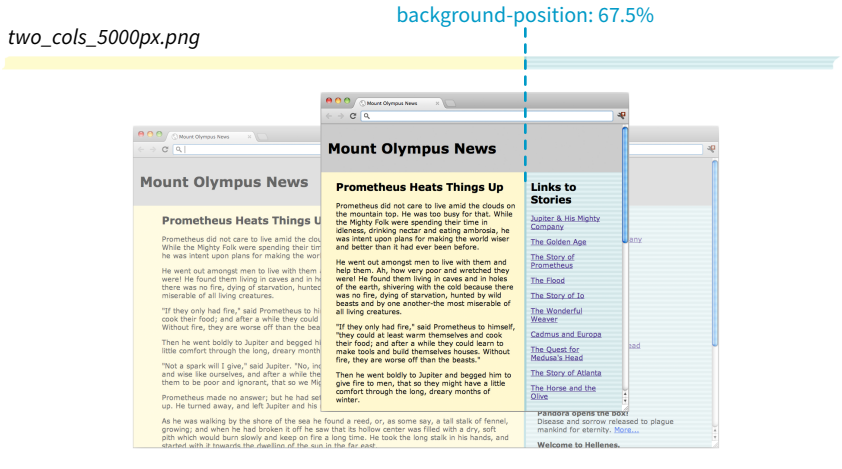


FIGURE N. The background image is anchored at the point between the two columns, so when the browser window gets larger or smaller, it is always in the right place. The graphic file is wide enough that there will be enough image to fill both columns, even on the widest of browsers.

Three Faux Columns

Well, that works for two columns, but what about three? It is certainly possible with the help of multiple background images.

Fundamentally, the process is the same as the one we just saw: position a really wide background graphic proportionally in a container **div**. But for three columns, you position two background images. One image provides the color band for the left column, and the remaining right portion is transparent. A second image provides the color for the right column, with its left portion remaining transparent (FIGURE O). The background color of the page shows through the transparent areas and provides the color for the middle column.

The markup requires a container (**#wrapper**) wrapped around the columned portion of the layout:

```
<div id="wrapper">
  <main></main>
  <div id="news"></div>
  <div id="links"></div>
</div>
```

The left-column graphic goes in **#wrapper**, positioned at the point between the left and center columns (26.25% for the example in FIGURE O). The right-column graphic is positioned between the center and right columns (73.75%). When the browser window resizes, the background images stay put at their proper point between columns, and the background color fills in the space in between.

```
#wrapper {
  background-image: url(left_column_bkgd.png), url(right_column_bkgd.
png);
  background-repeat: repeat-y, repeat-y;
  background-position: 26.25%, 73.75%;
}
```

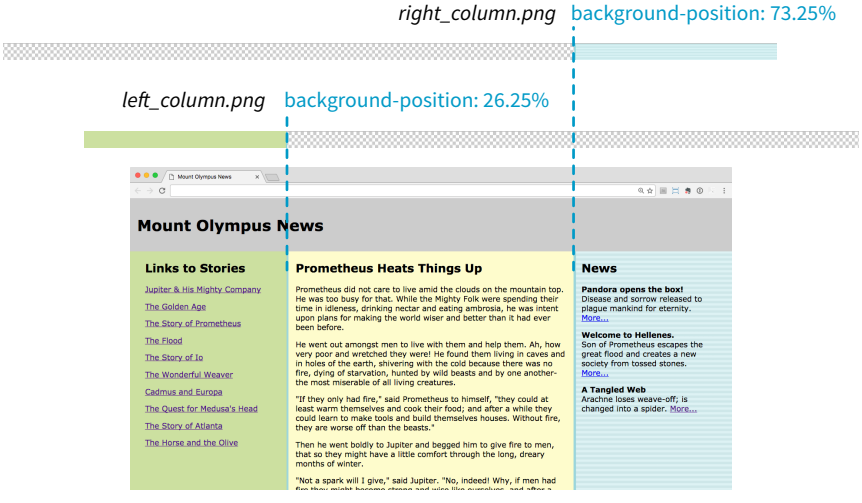


FIGURE O. Faux columns for a fluid, three-column layout.

BROWSER SUPPORT NOTE

Multiple background images do not work on Internet Explorer 8 and earlier. If your site traffic demands that you support these old browsers, you can use Doug Bowman's "Liquid Bleach" trick from way, way back in 2004 (stopdesign.com/archive/2004/09/03/liquid-bleach.html).

A FEW LAST WORDS

That wraps up our tour of column layout techniques. Once you get the hang of floating elements, using margins to make room for them, and using negative margins to pull things to the left in the layout, you've pretty much got all the tools you need to make columns in fixed or fluid layouts using floats. Positioning is even more straightforward once you get comfortable with the position properties.

A few words of caution: make sure that your widths and margins do not exceed 100% of the viewport or other container. If you are using the **`content-box`** box-sizing model, you'll have to account for padding and borders in the total width as well, which may be a good motivation to set **`box-sizing`** to **`border-box`** for the whole page.

Finally, be careful when combining percentage and pixel or em measurements (for example, a 50% wide column with 20px margin). Browsers may round percentages differently, potentially causing "float drop" if they round up higher than the available width of the browser.

And again, the techniques in this article are antiquated and should be used only as fallbacks for browsers that don't support CSS Grid and Flexbox. One day, they will join tables and 1-pixel transparent GIFs (see **Note**) in the Museum of Hilarious Web Design Hacks.

NOTE

Before style sheets, one method to add space on a web page was to place a transparent 1 × 1 pixel GIF on the page with an **`img`** *element, then give it a width and height to hold that amount of space clear. Can you believe it?! Aren't you glad you're learning web design in the Age of CSS?*