



## Float: The Theory

By: [John Gallant](#)

This article is intended for people familiar with with css syntax and usage. If you are new to css, see Adrian Senior's excellent article, [CSS: An Introduction](#) (<http://www.communitymx.com/abstract.cfm?cid=EA87013F0E8B6A50>), in the articles section of CMX. Following that, you should read [Flowing And Positioning: Two Page Models](#) (<http://www.communitymx.com/abstract.cfm?cid=EA1A82A6F65A33F5>), in the same section, to better understand this article.

If you are like most people, the whole issue of Floats is rather intimidating. The theory is difficult to grasp, and on top of that, the bugs are rumored to be numerous and nasty. Well fear not, because we will walk through the concepts, some of the bugs, and the practical uses of floats, in nice easy stages. Be assured that you will suffer no ill effects. Rather, a vast new world of positioning will open up before you. Onward!

### Basic behaviors

Any element may be floated. Paragraphs, div's, lists, tables, and images can all be floated, and in fact even inline elements like "span" and "strong" can float just fine.

Any element that is declared to be a "float" is automatically made a "block level element," meaning it can have both a declared "width" and "height." In fact, floats are *required* to have a declared width. A very common mistake is to leave off this float width, and while some browsers let you get away with it, more standards compliant ones do not. Without that width, the float could either be as wide as the content within, or as wide as the float's container element. Browsers have yet to agree which is correct, and even the W3C is apparently waffling over this issue.

Floats are "**removed from the flow**," but unlike absolutely positioned elements (layers), floats do get positioned directly after the last block element that precedes them (just like block boxes). If the float is inside a "line box," the float's upper edge is placed level with the top of the line box. But other than that, floats are similar to absolute elements, in that ordinary block boxes totally ignore both floats and AP elements. Those static block boxes just keep "flowing" one after another as though the float were not there.

The following is minimal code for the example graphic:

```
<div style="float:left; width:40%;"><p>Float text<p></div>

<p>
Paragraph text outside the float
</p>
```

This floated box precedes the following block box, and lays over it, forcing the "line boxes" within that box to be shortened so they won't be covered by the float. This float has a left margin to show the overlap effect.

This is a "block box" and it ignores the previous float, which lays over it. Only the inline content you are reading is forced to stay out from behind the float. The borders and background of this rectangular box extend beneath the float.

You can get inline elements like text and images to go along the side of a float and continue below it, just like using the good old (but deprecated) "align=left" on an image. Floats, however, are much more versatile than that.

It is a common misconception that block elements following a float are made to run down the side of the float, but that is not correct. Actually, it is only the "line boxes" within those block elements that behave so. (Please see the first screen shot image.)

Remember that I mentioned that floats are removed from the document flow? The specs require that line boxes that **would** pass behind a float be **shortened** enough to keep the line box out in the open. That means text and inline images cannot ever be covered by a float, unless one or both are in separately positioned containers. That is, if a float were nested in an absolutely positioned element, and that AP element were made to overlap a paragraph, the line boxes in the paragraph would **not** be shortened by the float, due to their different contexts. Rather, that text would indeed be covered by the float.

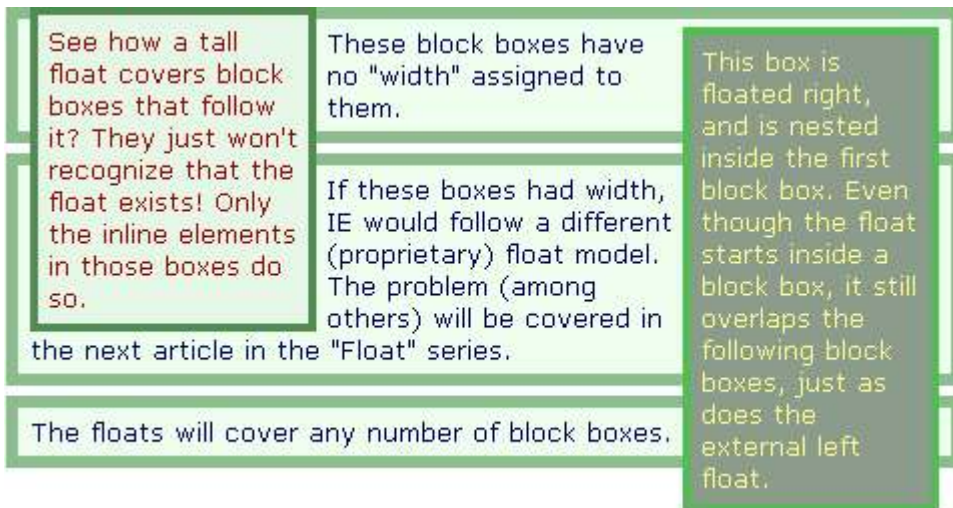
A common problem people have with floats is keeping text outside the float from touching the sides of the float. Putting left padding or margin on the paragraph following the float won't work, because those properties are applied over on the left side of the paragraph, *behind the float!* The proper way is to place margins on the float itself, thus convincing the line boxes that the float is "bigger" than it appears, and shortening them accordingly. Most floats typically get such margins just on the side nearest the text and the bottom. The float in the screenshot above has a small right margin.

Meanwhile, the block boxes following the float just get covered by the float (even though their content may have been moved aside). If the float is made very tall, it can drape over a large stack of block boxes, as shown in the second screen shot image.

```
<div style="float:left; width:30%;">
<p>Left float text</p></div>

<p> <!-- first block box -->
  <div style="float:right; width:150px;">
    <p>Right float text</p></div>
  Text following the right nested float
</p>

<p>Paragraph</p> <!-- second block box -->
<p>Paragraph</p> <!-- third block box -->
```



As seen above, another behavior of floats is their "direction". When a float is defined, it is always given a directional value of either "left" or "right," for example: `div {float: left;}`. These values simply move the float to the indicated side of the containing box.

Occasionally, float newbies will try to use float values such as `{float: up;}` or `{float: bottom;}`, but these values are not valid, and there are no plans to include them in the float specification. One can always hope...

## Where floats start

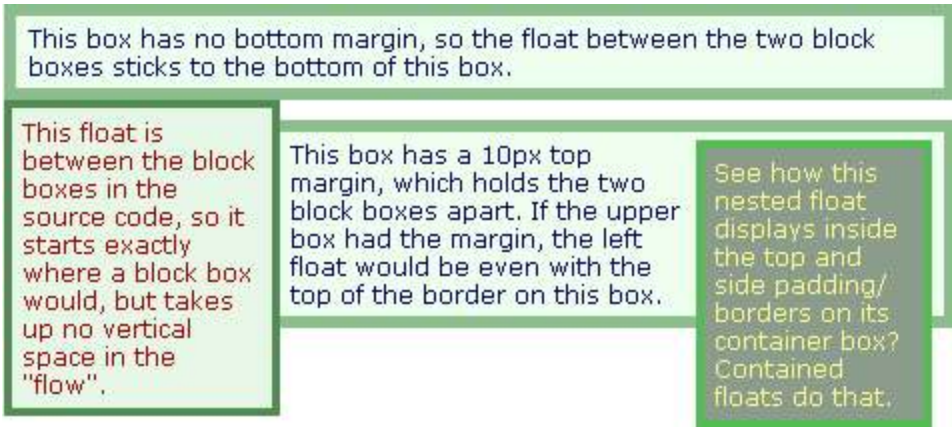
If a float comes between two block boxes, it has its upper edge placed against the lower edge of the preceding block box, and then it is moved over to the left (or right), until it strikes the side of the outer container element, often the `<body>` element.

If the float is nested **inside** the second block box, and that container box has no borders or padding, it appears just the same as when the float is between the block boxes. However, if those block boxes are held apart by margins, the float's top edge starts at the point where the margin of one box meets the other box. In the following screen shot image, the block boxes have top margins, but the left float's left margin **has been removed** to show its default (non-margined) behavior.

```
<p>Paragraph</p> <!-- first block box (complete) -->

  <div style="float:left; width:30%;">
  <p>Left float text</p></div>

<p> <!-- second block box -->
  <div style="float:right; width:150px;">
  <p>Right float text</p></div>
  Text following the right nested float
</p>
```

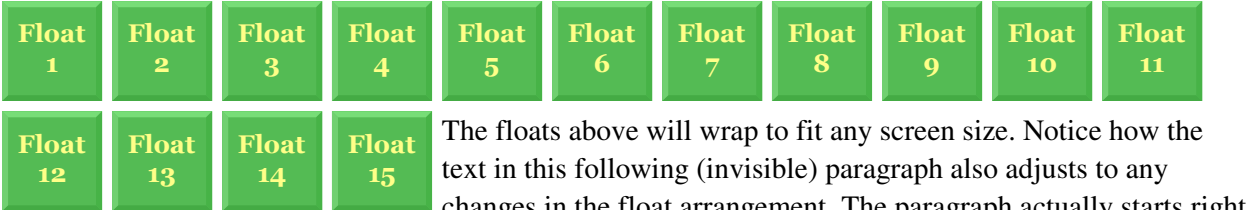


### Horizontal Float Stacking

If a left float is placed in the upper left corner of its container, and a similar float directly follows it, that float is placed at the upper right of the container, and then moved to the left, stopping against the right side of the first float.

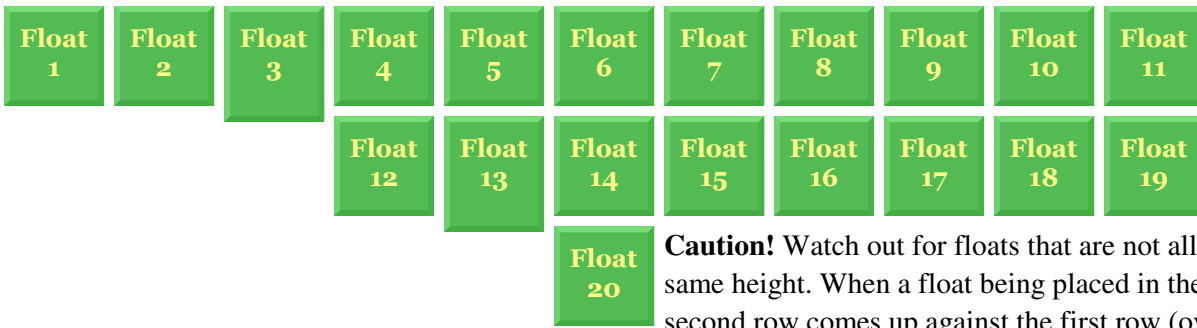
That last behavior lets us create a whole row of floats, each being placed to the right of the one before (or to the left of the one before if {float: right;} is used). Also, when there is not enough room for all the floats to fit in one row, the extra floats "wrap" down to another line, almost like inline elements do. This is very handy for making an array of clickable image "thumbnails," because the array will adjust to whatever screen size happens to be used, simply rewrapping as necessary.

Below is a line of colored left floats. The upper left float comes first in the source, and the lower right is last. They all have small margins just to look pretty. Try narrowing the window of your browser, and watch as the float line gets "wrapped" down to accommodate the window size.



If the float is given a "right" value instead of "left," the behaviors are exactly the same, but the float moves right rather than left, and subsequent floats are added to the left end of the row, not the right. In either case, wrapping is similar, unless float bugs cause differences.

Think of it this way. A float in a container is first moved up to the topmost space it can fit into (on the side away from its "direction"), then is moved in the designated float direction until it bumps into the side of the container, or another float. Each succeeding float does the same thing, until one of the floats can't find a space wide enough on the level of the others. Instead it is forced to stop against the bottom of the other floats, and then slide to the side just like before.



**Caution!** Watch out for floats that are not all the same height. When a float being placed in the second row comes up against the first row (over

to the right), and it tries to slide sideways (to the left), it gets "stuck" on the first taller float it encounters. With a large number of irregular floats this can get really ugly. The above batch of floats contains a couple of "joker" floats that are slightly taller than the others.

Test this behavior by narrowing your browser to different screen widths (browser abuse).

Whenever this technique is used to make a large thumbnail array, care must be taken to insure all the floats are the same height, or the page might easily resemble a train wreck.

All the live demos in this article have been hacked to account for the faulty IE5.x/win box model. This was necessary due to the borders and padding applied to the boxes in those demos. See [The Box Model Problem](http://www.communitymx.com/abstract.cfm?cid=E0989953B6F20B41) (<http://www.communitymx.com/abstract.cfm?cid=E0989953B6F20B41>) in the CSS article section for more on this topic.

## Opposing Floats

There's no reason why two successive floats can't have different directions. If a container starts its content with two floats, one **left** and the other **right**, and there is room for both to fit side by side, then they will indeed display that way. Any extra space will form a gap between the floats.

But what if there is not enough room for both? Then the float that comes last will be forced below the first float, although the floats will still remain on opposite sides of the container. Again, narrow the browser to see opposing float wrapping, or the lack of it in the case of percentage sized floats.

First, the left float precedes the right float:

This float has `width: 250px;` and is floated to the **left**. It **precedes** the following right float.

This float has `width: 250px;` and is floated to the **right**. It **follows** the preceding left float.

Then the reverse:

This float has `width: 250px;` and is floated to the **left**. It **follows** the preceding right float.

This float has `width: 250px;` and is floated to the **right**. It **precedes** the following left float.

Now percentage sized widths:

This float has `width: 44%;` and is floated to the **left**.

This float has `width: 44%;` and is floated to the **right**.

This simple arrangement mostly works well, but some browsers don't handle more complex (multiple) oppositions very well yet. Anything of the sort must be rigorously tested in all target browsers.

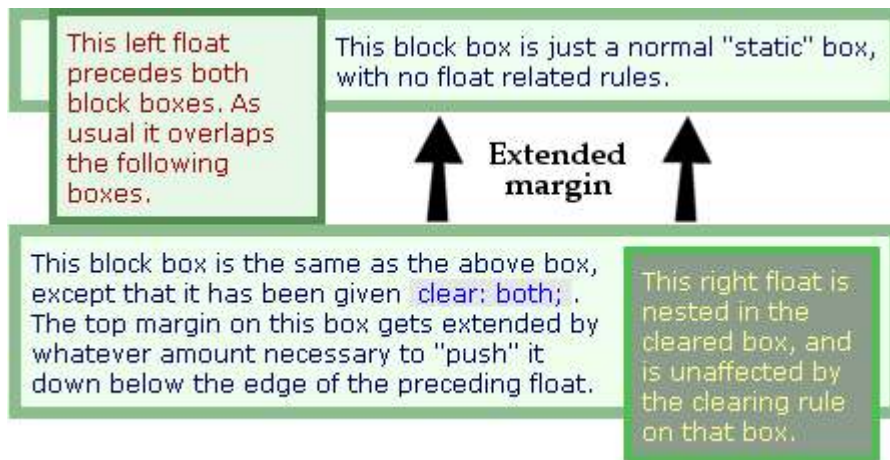
There is one major flaw in the above demo, in that Internet Explorer 5.x/win fails to make a following left float wrap below a preceding right float, under any circumstances, *unless* the floats are contained in a width defined block element. The above demo has such an element enclosing it. It's just one of many IE float bugs that are "fixed" by putting a dimension on the float container.

## Clearing Floats

Here we come to the trickiest and most misunderstood part of floating.

Think back to what was said about static boxes following floats. Those boxes just ignore the float, and display up against the previous static boxes. But let's say you give that following box the clear property, `{clear: both;}`. What this does is extend the margin on the top of the cleared box, pushing it down until it "clears" the bottom of the float. In other words, the top margin on the cleared box (no matter what it may have been set to), is increased **by the browser**, to whatever length is necessary to keep the cleared box below the float.

So in effect, such a cleared box cannot be at the same horizontal level as a preceding float. It must appear just below that level. The image shows how this might look.



Remember, floats are not actually contained within a block box, even though they may be nested there in the source code. Sure, the float's screen starting point is determined by its nested location, but after that it just drapes over the containing box, as do absolute elements. Only line boxes containing inline elements (like text) will care where the float is displayed.

The standard method of making an outer container *appear* to "enclose" a nested float is to place a **complete "cleared" element** last in the container.

```
<p> <!-- float container -->

  <div style="float:left; width:30%;"><p>Some content</p></div>

  Text in the float containing paragraph outside the float

<div style="clear:both;"></div>
</p>
```

Since that div is not floated, the container must recognize it and enclose it, and because of that top margin (added by the browser because of the "clear" property), the div "pulls" the bottom edge of the container down below the bottom edge of the float.

Sure, that is a weird way to do it, but that's what the specs say. However, there is talk at the W3C about adding a rule that would make a container "auto-enclose" a float. It sure would be simpler.

In fact, both Explorer and Opera 7 already do "auto-enclose" nested floats, in violation of the specs. Opera 6 did not do this, so it seems Opera is following Microsoft's non-standard lead.

It is also possible to use `{clear: left;}` or `{clear: right;}`. This allows an element to clear a left float but not a right, or the opposite. You will have to get pretty fancy to need this kind of control, however.

In general, clearing works well, but can sometimes induce minor and not so minor bugs, which are almost entirely confined to Internet Explorer.

## Coming Soon

Next in this series we will discuss some of the more serious bugs you might encounter when using floats (guess where), and ways to counter these bugs.

### Keywords

float, floats, CSS, block level element, line boxes, containing element, clearing floats,