

## Does CSS float?

A review of the behavior, use, and support of the CSS float property

Skill Level: Intermediate

[Michael Russell \(MikeRussell@VickiFox.com\)](mailto:MikeRussell@VickiFox.com)

Systems Architect

Vicki Fox Productions, Inc.

05 Mar 2008

The CSS float property is a popular tool in a Web designer's toolbox for page layout, but it is often poorly understood and hampered by inconsistent browser implementations. This article explores the float property and its common uses, developer tool issues, and browser inconsistencies.

From the earliest days of illuminated human writing, we see the practice of flowing text around images.

Early pioneers of the Web recognized the aesthetics and space savings of flowed text and included a "float" capability in Netscape 1.0 and updates to HTML 2.0. The ability to float elements has become a fundamental property of the HTML standards and the Cascading Style Sheet (CSS) specifications.

While the definition of the "float" property is straightforward, the implementation and use of the property has resulted in many Web page designers wasting time trying to get the page to behave as desired.

In this article, I explore the following aspects of the "float" property:

- Definition and conceptual model
- Common uses
- Developer tool issues

- Browser issues

## What is the "float" property?

The CSS2 Specification provides the definitive definition and behavior rules for the float property. In the following subsections, I expand on the CSS2 description found in Section 9.5, Visual Formatting Model (see [Resources](#) for a link).

### Definition

The float property specifies whether a box should float to the left or to the right. The property may be set for any element that generates boxes that are not absolutely positioned.

The property can have the following values:

- **inherit** - The element assumes the float property of the enclosing element. This is the default value.
- **left** - The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top. This flow is subject to an element with the "clear" property. The "display" property is ignored, except if the value is "none."
- **right** - The element generates a block box that is floated to the right. Content flows on the left side of the box, starting at the top. This flow is subject to an element with the "clear" property. The "display" property is ignored, except if the value is "none."
- **none** - The box is not floated.

### Behavior

The following rules govern the float property. I have only specified the left float behavior. To obtain the right float behavior, simply swap the direction (that is, left becomes right).

The specification defines the behavior in terms of box edges. The following is a reminder of the four different box edges in the CSS box model:

- The **content edge** or **inner edge** surrounds the rendered content of the element. This is the edge closest to the content.
- The **padding edge** includes the padding applied to the element. The padding edge defines the edges of the containing block.

- The **border edge** includes the border applied to the element.
- The **margin edge** or **outer edge** includes the margin applied to the element.

**Containing block horizontal constraint.** A left float box's left margin edge may not be to the left of the containing block's left padding edge.

**Containing block vertical constraint.** A left float box's top margin edge may not be higher than the containing block's top padding edge.

**Previous float constraint.** If the current box is a left float box and a previous box is a left float box then the current box's left margin edge must be to the right of the previous box's right margin edge, or the current box's top margin edge must be lower than the previous box's bottom margin edge.

**No overlap constraint.** A left float box's right margin edge may not be to the right of any right float box's left margin edge that shares the same horizontal space.

**Previous element vertical constraint.** A left float box's top margin edge may not be higher than the top margin edge of any block or floated box generated by an earlier element in the source document.

**Line-box vertical constraint.** A left float box's top margin edge may not be higher than the top padding edge of any line-box containing a box generated by an earlier element in the source document. A line-box is an imaginary rectangle that contains all the inline boxes that make up a line in the containing block-level element. Its height (top padding edge) is the height of the tallest line-box.

**Opposite horizontal margin constraint.** A left float box that has another left float box to its left may not have its right margin edge to the right of the containing block's right padding edge. That is, a left float box may not "stick out" past the right edge of the containing block, unless it is already positioned as far to the left as possible.

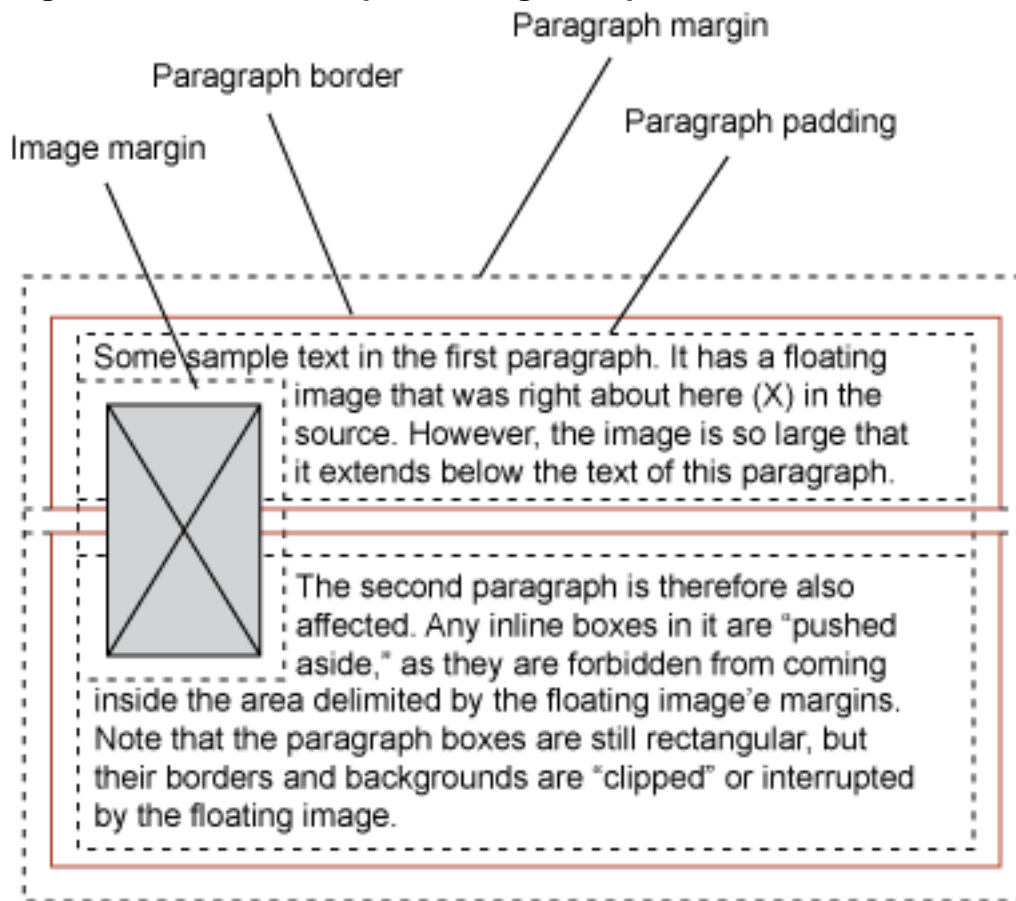
**Optimum vertical constraint.** A float box must be placed as high as possible within its containing block.

**Optimum horizontal constraint.** A left float box must be put as far to the left as possible within its containing block.

**Position precedence rule.** When resolving the float box position, the optimum vertical constraint takes precedence over the optimum horizontal constraint. That is, favor moving the float box to the top before moving to the side.

Figure 1 illustrates how the behavior rules would position an image with a margin relative to a text block.

**Figure 1. Left float box positioning example**



## Additional considerations

One area where new Web designers may get confused is determining where to put the floated element within the source document. The designer can use these rule-of-thumb guides to help resolve the confusion.

**Positioning.** The browser determines a floated element's vertical position from the element's location in the "normal flow" of the document. The normal flow is how the document would appear if the browser ignored positioning properties. The floated element is then taken out of the flow and moved as far left (or right) as possible within its containing block.

**Inline becomes block.** A floated element becomes a block box for page formatting. It is equivalent to specifying the `display: block` property.

**Required width.** A floated element should have a width specified. The CSS2 Specification states that the element should have an explicit width using the "width" property or an implicit width computed from contained sub-elements, such as an image. Images have an implicit width as part of the image's properties. If a width is

not specified, then the results are unpredictable.

## What is the "clear" property?

Subsequent elements following a floated element will wrap around the floated element. In general, for text, this is the desired effect. However, if you are using float for layout, you may want to stop the wrapping behavior. To stop this behavior, the subsequent element needs to specify the "clear" property.

The CSS2 Specification provides the definitive definition and behavior rules for the clear property. In the following subsections, I expand on the CSS2 description found in Section 9.5, Visual Formatting Model (see [Resources](#)).

### Definition

The *clear* property indicates which side or sides of an element's box may not be adjacent to an earlier float box. If the element itself has floating descendants, then the clear property has no effect on those descendants.

The clear property only applies to block-level elements. This includes those elements made into a block-level element by using the float property.

The property can have the following values:

- **inherit** - The element assumes the clear property of the enclosing element. This is the default value.
- **left** - Increase the generated box's top margin so that the top border edge is below the bottom margin edge of any left float boxes generated from previous source elements.
- **right** - Increase the generated box's top margin so that the top border edge is below the bottom margin edge of any right float boxes generated from previous source elements.
- **both** - Move the generated box below all floating boxes generated from previous source elements.
- **none** - Apply no constraint on the box's position with respect to previous floated elements.

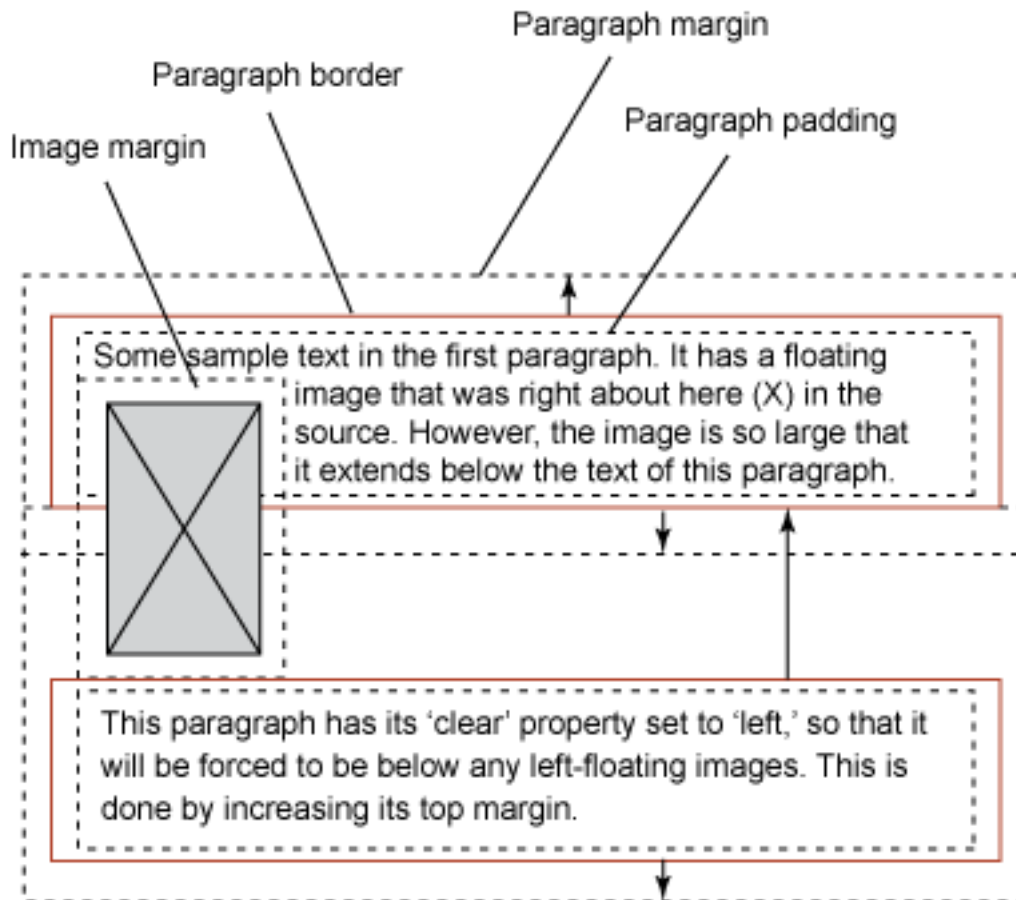
### Behavior

The behavior of the clear property is in essence a modification of the rules for positioning a floated element. In the Specification, the behavior is treated as an extra constraint.

**Clear constraint.** The top margin edge of the float box must be below the bottom margin edge of all earlier left float boxes (in the case of "clear:left"), or all earlier right float boxes (in the case of "clear:right"), or both (in the case of "clear:both").

Figure 2 shows an example of the use of the clear property. Both paragraphs have the property "clear:left". This has no effect on the first paragraph or the float box defined within the first paragraph. The clear causes the second paragraph to be positioned below the float. Notice how the top margin of the second paragraph is extended vertically to meet the bottom margin of the previous paragraph.

**Figure 2. Clear positioning example**

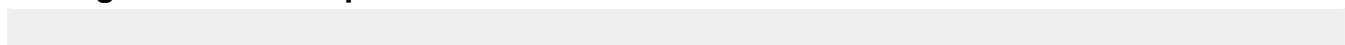


**Additional considerations**

A common difficulty associated with the clear property is that its use requires adding extra markup in the document on subsequent elements.

Listing 1 shows how adding an empty division adds non-content markup to the HTML document.

**Listing 1. Extra markup to handle "clear"**



```
<div class="myFloatClass">
  <p>myFloatClass has float:left specified</p>
</div>
<div class="myClearClass" />
  <!--myClearClass has clear:left specified-->
```

Several authors have proposed various techniques to eliminate the need to add extra markup to the HTML document. These techniques include:

- Add "float" to the container element.
- Add "overflow:hidden" to the container element.
- Add a CSS2 ":after" pseudo-class on the container element .

**Float Container Technique.** This is the approach often seen in page layouts and horizontal navigation menus—put a float element within a float container. The float container will expand to encompass all interior float elements. This approach requires correctly setting the width property of the container, with 100% being a common setting.

This technique has several downsides. First is the difficulty of setting the width, or the fact that using 100% may cause conflict with padding. Secondly, Internet Explorer V6 may add extra bottom margin. Finally, deep nesting of float boxes may cause unpredictable behavior in some browsers.

Listing 2 shows the CSS code for the float container technique.

### Listing 2. Float container technique

```
.myFloatClass {
  float: left;
  width: 100%;
}
```

**Hidden Overflow Container Technique.** This approach is seldom used. It involves taking advantage of the "overflow" property and default "clip" property behaviors—an overflow element will expand to the size of all contained sub-elements, which includes floated elements.

This technique has several downsides. First, the use of overflow may affect fluid height for the container. Secondly, the overflow requires triggering the Internet Explorer V6 "hasLayout" property.

Listing 3 shows the CSS code for the hidden overflow container technique.

### Listing 3. Hidden overflow container technique

```
.myFloatClass {
  overflow: hidden;
  height: 1%;
  /* Or zoom:1 to trigger IE's hasLayout */
}
```

#### Share this...



Digg  
this  
story



Post  
to  
del.icio.us



Slashdot  
it!

**:after pseudo-class technique.** This approach is rarely used. It involves using a CSS pseudo-class to generate content after the container. Use the `:after` pseudo-class to append to the resulting HTML document a new element that has the `clear` property specified.

This technique is the least favorable and has the least browser support. The `:after` pseudo-class is not supported by Internet Explorer V7 or earlier. But most importantly, this technique inserts meaningless content into the resulting HTML document.

Listing 4 shows the CSS code for the `:after` pseudo-class technique.

### Listing 4. `:after` pseudo-class technique

```
.myFloatClass {
  height: 1%;
  /* Or zoom:1 to trigger IE's hasLayout */
}
.myFloatClass:after {
  content: ".";
  display: block;
  height: 0;
```

```
clear: both;
visibility: hidden;
}
```

## What are some uses for "float"?

The float property has become a frequently used tool in the Web designer's toolbox. The number of uses is as wide as the designer's imagination. The following are some of the commonly encountered uses of float.

The [Resources](#) section lists several books and Web sites that provide details on how to use float to achieve these effects.

**Anchoring images.** Moving an image to the left or right is only half the story. Because the float is effective at the point where the element is included in the source document, the float is anchored at that point. This is especially important with text documents where you want the image to stay relative to the text that describes the image.

**Adding a caption.** Often it is desirable to add a caption, such as "Figure 1 Example image," to an image. By putting the image within another container, such as a division (<div>) block and floating the division block, you can include a caption with the image.

**Large capital letter.** You can recreate the classical book style of a large capital letter appearing in the upper left corner of a block of text. You cannot just change the font size, because the letter will extend above the rest of the text. Instead, you want the letter to extend downward into the text and the rest of the text to wrap around the letter.

**Inline lists.** A combination of float and display:inline are used to convert unordered lists (<ul>) into horizontal menus or navigation tabs. By representing menus as unordered lists, a menu will appear as a distinct list of options on Web browsers that do not render advanced or graphical layout (such as text-only browsers).

**Multicolumn page layouts.** Using tables to lay out columns on a Web page often results in very difficult-to-maintain Web pages due to a lot of markup in the source document that contributes nothing to the content. With the growth of Web standards-compliant browsers, it has become easier to use divisions and CSS to achieve similar layout with far less markup in the source document. The float property is one way to get division blocks to position similar to table cells.

**Flexible gallery pages.** Traditionally, tables are used to lay out image galleries. The problem with tables is that they are not fluid and do not respond well when the user resizes the display window. By using floated images instead, the gallery looks like a table layout, but the layout becomes narrow or wide in response to a user resizing the window. For example, showing 12 images on a flexible gallery page can result in the following display combinations: 1x12, 2x6, 3x4, 4x3, 6x2, and 12x1.

## Are there developer tool considerations?

As every developer knows, not all HTML/CSS WYSIWYG editors are equal. Each editor has its own quirks, levels of Web standards compliance, and choice of page rendering engines.

My experience using popular WYSIWYG editors, such as IBM® Rational® Software Architect V7, Adobe Dreamweaver CS3, and Genuitec MyEclipse V6, is that they are useful to get a page started, but often force working at the source code level for fine-tuning a page. It is vital to test the page in the target browsers. Many editors include add-on support for various browsers making it possible to test the page by launching the browser from the editor.

If you are in a position to evaluate a WYSIWYG editor, I suggest subjecting the editor to the Acid2 Browser Test (see [Resources](#)). The Web Standards Project developed the Acid2 Browser Test to test the level of Web standards compliance among browser vendors. However, you can also use the Acid2 Browser Test source to test the Web standards compliance of your candidate HTML/CSS WYSIWYG editor.

## Are there browser considerations?

Web designers have identified a number of non-standard behaviors in popular Web browsers.

Various browsers implement nested floated elements differently. As a result, you may want to avoid using deep nesting of floated elements for positioning. In these cases, the better option may be to use relative and absolute positioning properties.

**Internet Explorer.** The [Position Is Everything](#) Web site maintains a catalog of Internet Explorer bugs and the hacks to fix them. See the [Explorer Exposed page](#).

**Firefox.** The main bug related to float I have found in Firefox is the Top Gap Bug. This bug often appears in multicolumn layouts where the columns are floated but the header section is not. An element in one of the columns with a top margin will see that margin pushed above the header, resulting in a gap between the top of the window and the top of the header section. The fix consists of putting something above the floated columns. One approach is to add an element to the bottom of the header that has a clear:both property. Another approach is to float the header, as well.

**Opera.** The [Position Is Everything](#) Web site maintains a catalog of Opera bugs and the hacks to fix them. See the [Opera Omnibus page](#).

## Summary

The ability to float elements on a page has existed from the early days of Web browser technology. Modern Web design is using the float property more and more to obtain page layouts previously done using tables. Despite the simple definition and behavior of the float property, browser inconsistencies have made using the property difficult. This article has only scratched the surface of the float property. For more, check out the [Resources](#) section, and of course, don't be afraid to play around on your own.

# Resources

## Learn

- Figure 1 and Figure 2 are copied from the [Cascading Style Sheets, Level 2, CSS2 Specification](#) document. Permission to use the figures is granted within the [W3C Copyright statement](#).
- Eric Meyer's online article, "[Containing Floats](#)," describes the relationship of float and normal flow.
- A helpful desk reference to CSS is Eric Meyer's [CSS: The Definitive Guide \(3rd ed\)](#), published by O'Reilly.
- Christopher Schmitt's [CSS Cookbook \(2nd ed\)](#) published by O'Reilly provides a number of samples of CSS in use.
- Several tutorials on the use of float for image and page layout are at the Max Design site, [Floatutorial](#).
- Robert Nyman's blog entry, [How to Clear CSS Floats Without Extra Markup - Different Techniques Explained](#), shows three ways to perform "clear" without having to create extra markup in the Web page.
- [Cascading Style Sheets, Level 2, CSS2 Specification](#), published by W3C, is the authority on the specified behavior of CSS properties. [Section 9.5](#) covers floats.
- [The Web Standards Project \(WaSP\)](#) promotes the acceptance of Web standards.
- Use the [Acid2 test](#) to test Web browser compliance with Web standards.
- The [Position Is Everything](#) site maintains a large wealth of information about browser incompatibilities, bugs, and hacks.
- Michael Russell's article, "[Compare Web site appearance and functionality](#)," discusses the advantages of using Web standards instead of browser extensions.
- Read "[Cross-browser Web application testing made easy](#)."
- The [Hoverbox Image Gallery](#) deals with float quite cleverly, and has been [covered a few times in developerWorks](#).
- Get the resources you need to advance your skills in the [Web development area on developerWorks](#).

## Get products and technologies

- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

## Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).

## About the author

### Michael Russell

Michael Russell has a bachelor's degree in physics and a master's degree in computer science. He was a logistics engineer, a technical services manager, and a certified IT architect at IBM for nearly 14 years. Michael is currently a systems architect in the defense aerospace (simulation and logistics) industry. He has experience in Windows, Unix, and OS/400 environments and uses Web technology for entertainment through his company's Web site.

## Trademarks