How to get Cross Browser Compatibility Every Time

Summary

A quick summary of the important points which would be helpful in achieving cross browser compatibility.

1. Always use strict doctype and standards-compliant HTML/CSS
2. Always use a reset at the start of your css
3. Use -moz-opacity:0.99 on text elements to clean up rendering in Firefox, and text-shadow: #000 0 0 0 in Safari
4. Never resize images in the CSS or HTML
5. Check font rendering in every browser. Don't use Lucida
6. Size text as a % in the body, and as em's throughout
7. All layout divs that are floated should include display:inline and overflow:hidden
8. Containers should have overflow:auto and trigger hasLayout via a width or height
9. Don't use any fancy CSS3 selectors
10. Don't use transparent PNG's unless you have loaded the alpha

Element Inconsistencies

Every browser renders particular elements differently - different amounts of padding, margins, borders etc. This means your site can look different in every browser if you use default styles.

Solution

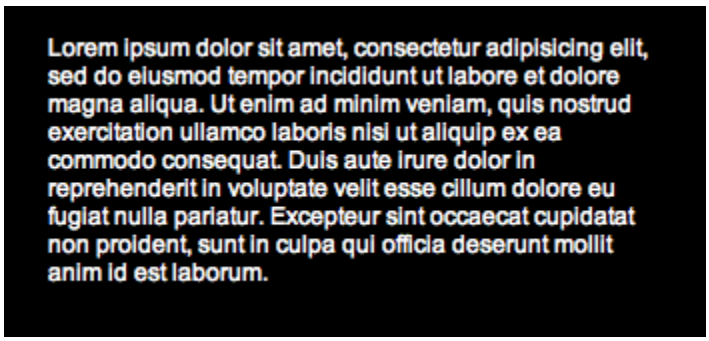
The first thing you should do, which most people most likely know about, is to reset your styles. A reset is some css you place at the start of your styles that removes the padding, margin, border and other inconsistencies from elements, and rebuilds them in a standard way. Download the reset.css and place the code at the start of your styles.

Image Rendering

IE6 and IE7 both render resized images extremely badly. You should ever resize an image smaller or larger, in either the CSS or HTML, as it will look ugly in the browser.

Font Rendering

Not all issues were with IE. Safari 3+ has an issue with the way it renders light type on a dark background. There's a way to make it appear lighter.Given below is an example of standard white on black text rendering in Safari 3.1:



Solution

Easy fix. Add the following to your code.

p { text-shadow: #000 0 0 0; }

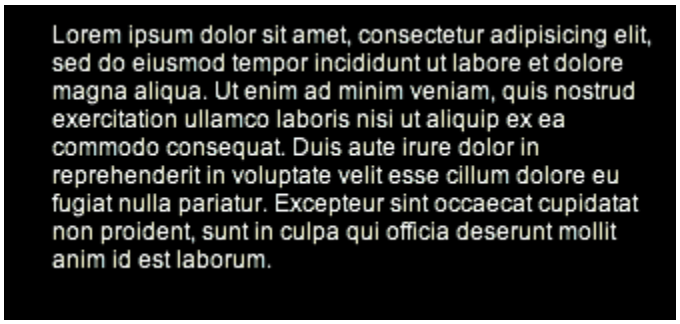
Where #000 is your background colour. We will probably have to be more specific with the elements you select. It wouldn't be advisable using this fix on the body tag. Other elements you might need to fix are the li, dt, dd, blockquote etc. Use this on any text element you want to appear 'thinner'.

To make this fix in Firefox, you use the opacity fix:

p { -moz-opacity: 0.99; }

You need to be careful with this fix, as it will break any Flash element that it touches in Firefox. There appears to be no workaround for it.

The type will now be rendered like this:



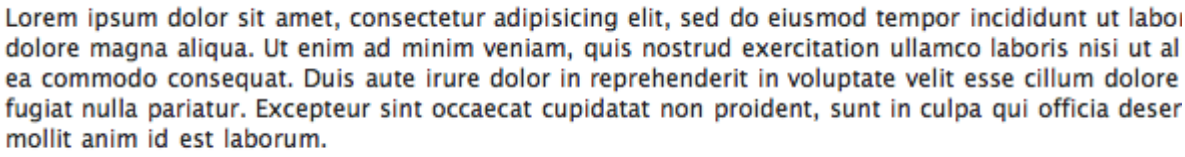
Font Selection

There are common web fonts that we use - Arial, Georgia, Verdana etc. But there are some fonts that are common to both PC and Mac and can be used - Century Gothic, Arial Narrow etc. However, different browsers and OS's render type different, and you need to be aware of these differences, as your site could look dam ugly if you use the wrong font.

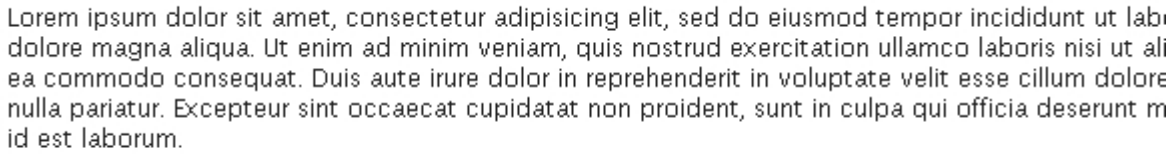
Solution

The font you choose is ultimately up to you. As long as it's a safe font you will be fine. However you should be aware of these rendering issues. One font you should probably not use is Lucida Grande/Sans. It renders awful in IE.

Given below is the rendering in Safari (Mac):



The above text is looking proper in Safari but doesn't look nice in Internet Explorer.It looks like the following in Internet Explorer:



Some people have stated that Lucida Sans will look fine in IE with ClearType, and others have said that it still looks bad. A smart alternative is to just not use Lucida Sans as your 'fallback font', and instead use Arial or another san-serif font. It's really up to personal choice.

Font Sizing

The ability to resize text differs amongst browsers and OS's. IE won't resize text thats set in pixels. If we set all text in em's, IE will exaggerate the text sizes when resized.

Solution

The best reference for font sizing is the article How to Size Type in CSS by Richard Rutter. The results - You need to specify the size as a percentage in the body element, and then size it in em's through the rest of the sheet. For line height, you need to define it in em's, rather than pixels, for consistent rendering.

One thing to remember is that the default font size is 16px. So to resize our type to 12px we use:

body { font-size: 75%; line-height: 1.5em; }

Now you can size your type in em's like so:

h1 { font-size: 3em; }

Another suggested technique is to resize it down to 10px, which makes it easier to size upwards in em's. For example.

body { font-size: 62.5%; line-height: 1.5em; } h1 { font-size: 1.8em; /* 18px */ } p { font-size: 1.2em; /* 12px */ }

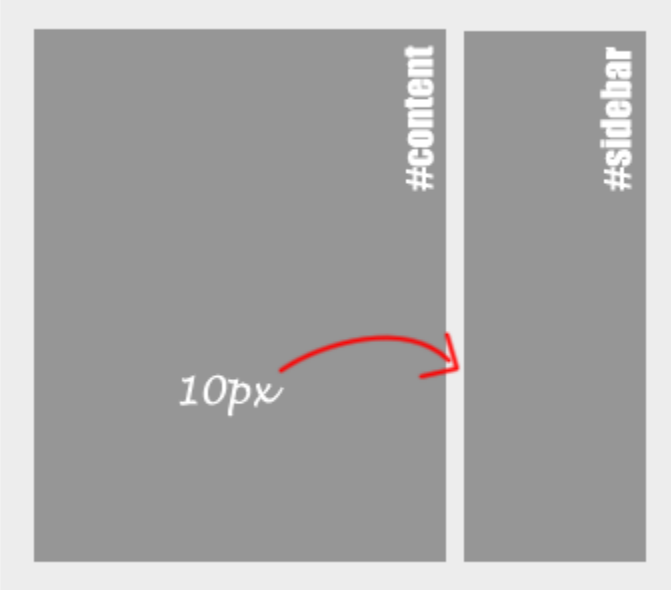
There is an issue with this however, if people use small font option in Internet Explorer, it could be so small that it's unreadable. It's only a small chance. But worth noting.

Double Margins on Floats

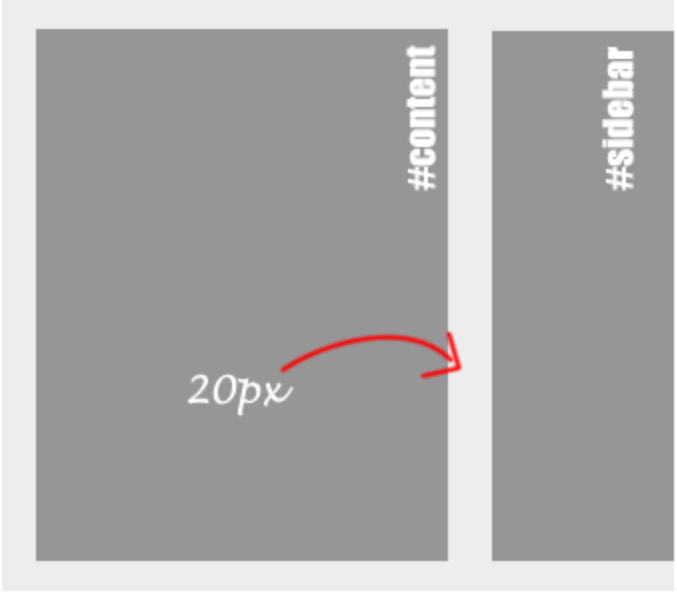
We create CSS layouts by floating div's up against each other, like some form of horizontal tetris. When it reaches the end of a row, it drops down to the next. The common method for creating a grid is this:

#content { float:right; width: 300px; margin-right: 10px; } #sidebar { float:right; width: 100px; }

This is what it should look like in a browser:



This is what IE does to it:



The margin width is doubled. **Any margin that is on the same side as the float will be doubled**. This means the element on the left will have that margin stretched out to 20px, which will probably break the layout.

Solution

To fix it, all you need to do is put display:inline on your layout divs.

#content { float:right; width: 300px; margin-right: 10px; **display:inline;** } #sidebar { float:right; width: 100px; **display:inline;** }

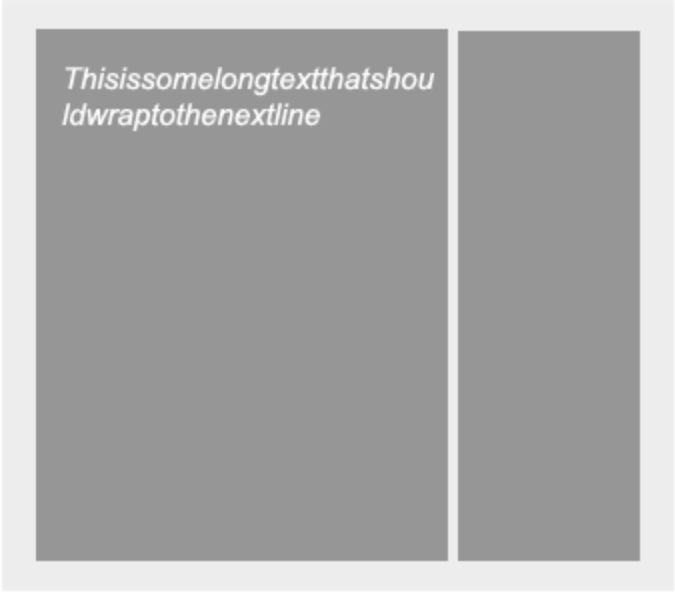
Now that margin bug is all fixed. Even though it only effects margins on the same side as the float, it can still be useful to include this fix. You never know when you might use a margin on the float side. It won't effect any other browser, so you can use it safely.

Expanding Box

The expanding box problem is a big issue with many css layouts. When you have a standard layout, with floats sitting next to each other, with set widths, but an image or long string of text is longer than this width, the layout will break in IE6. Take a look at this example:

#content { float:left; width: 300px; margin-right: 10px; display:inline; } #sidebar { float:left; width: 100px; display:inline; }

This will render like in this most browsers:



However, in IE6, it will break like so:

Solution

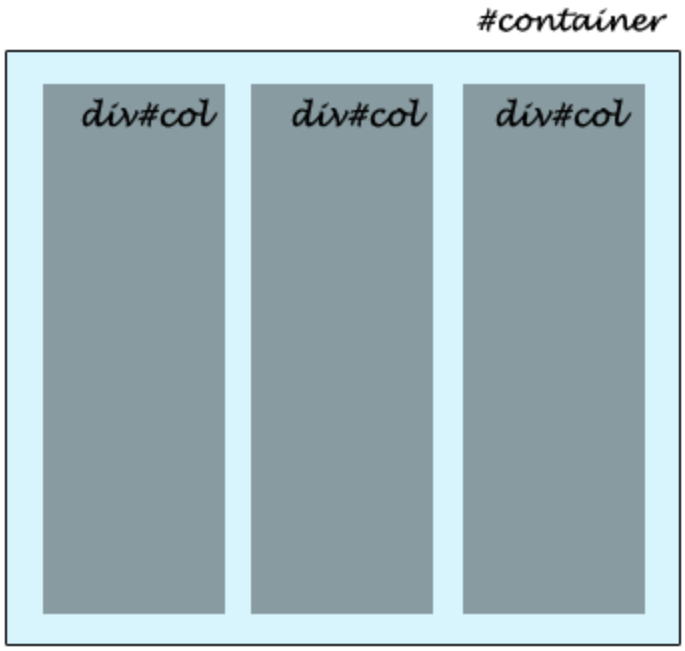
The solution is to use the overflow property. If you place overflow:hidden; into the layout divs, the layout won't break in IE6.

#content { float:left; width: 300px; margin-right: 10px; display:inline; overflow:hidden; } #sidebar { float:left; width: 100px; display:inline; overflow:hidden; }

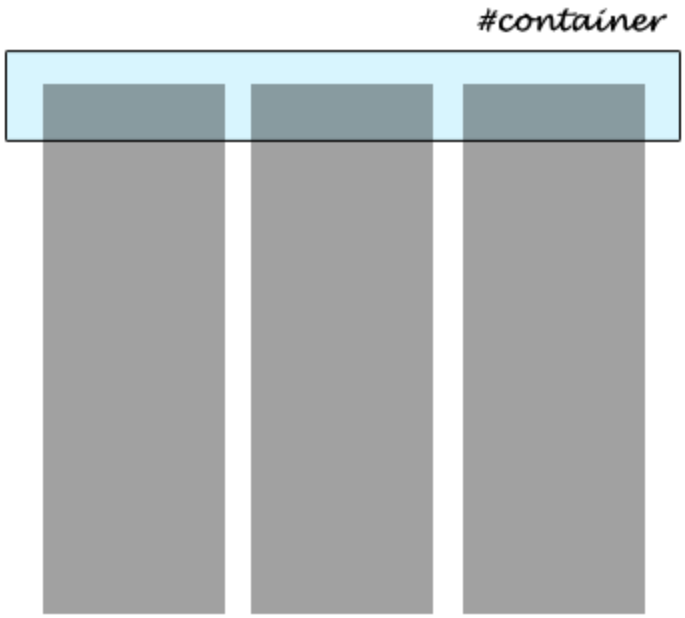
This could, however, cause some issues with layouts depending on how complex they are. It also won't cause the text to wrap, and it will just remain hidden. It's unlikely you'll have a string long enough to break a layout, but it could happen. Another option is to use word-wrap: break-word; in an IE specific stylesheet. This won't effect images though, so you'll have to make a choice about with method is appropriate for your situation.

Clearing Floats

What we mean when we talk about clearing floats, is when a container or wrapper div doesn't correctly wrap around the containing divs. Take a look at this example:



See how the container is correctly containing the div's? This is what should happen. However, sometimes it does not clear correctly, like so:



The container isn't correctly wrapping around it. You probably won't notice this until you try and put a background on your container.

Solution

There are a few different solutions for clearing. The best solution however, is a simple **overflow:auto or overflow:hidden** in your container

#container { width: 966px; margin: 0 auto; overflow:auto; }

You need to keep in mind that overflow:auto might cause some issues in Firefox. If you do have any issues, use overflow:hidden instead. If this is still causing issues, take a look at some other forms of clearing floats. Apart from just adding this, you'll need to make sure hasLayout is triggered in IE6. You can do this by specifying a width or height. If you don't have a width in your container, you can use height:1% to trigger it, or zoom:1; if you can't afford to give it a height

CSS Selectors

Although we would like to use all the brand spanking new CSS3 selectors, IE6 doesn't support all of the major ones. There are also still some CSS2 child and sibling selectors you simply shouldn't use if you want to support IE6:

- E > F
- E + F
- E ~ F
- :root
- :last-child
- :only-child
- :nth-child()
- :nth-last-child()

- :first-of-type
- :last-of-type
- :only-of-type
- :nth-of-type()
- :nth-last-of-type()
- :empty
- :not()
- :target
- :enable
- :disabled
- :checked
- Any of the E[attribute] Selectors
- :first-child
- :lang()
- :before
- ::before
- :after
- ::after

Of course, a lot of these selectors aren't supported by even Firefox 3. For a full list of supported selectors, check out evotech.net's post on browser css selector support

Solution

Stick to your standard selectors. Nothing fancy. Only use E + F, E > F, E ~ F when it won't make a huge difference in IE. If you **really** need to use these selectors, you should look at using IE8.js which gives IE6 better selector support. However this will slow down your site. But we don't really care about IE6 users do we? :)

PNG Transparency

IE6 doesn't support alpha transparency in PNG's. This is possibly the most annoying bug/issue with IE.

Solution

There are a few solutions for this problem. You can either use AlphaImageLoader in an IE specific stylesheet, link to a behaviour file in an IE specific stylesheet or use JS to fix the issue. Not matter which way you choose, there is no way to have transparent repeating background images.

To use AlphaImageLoader, it's a little bit tricky. Add these properties to any png image you want to have transparency. (You need to put this code in an IE specific stylesheet if you want your CSS to validate)

.trans { background-image: none; filter:
progid:DXImageTransform.Microsoft.AlphaImageLoader(src='/images/transparent.png',
sizingMethod='image/scale/crop'); }

Let's say you've given all your png image tags a class "trans", you select them with your css and use the filter property. You need to create a 1x1 transparent png file and link to it in the src attribute. An even better way to do this is via a behaviour. It's similar to the AlphaImageLoader, except that you link to a behaviour script that triggers the transparency.

The third method is to use IE8.js which I mentioned earlier. It's even safer than the previous method, and you are very unlikely to run into any problems. Link to the script in your HTML, and it will make any png ending in -trans alpha-capable. eg. myimage-trans.png.

With all this methods, you need to be aware that you might still run into problems with transparent pngs. Make sure you test it in IE6 extensively