

Creating a CSS driven menu system – Part 1

How many times do we see in forum pages the cry; "I've tried using Suckerfish...", "I've started with Suckerfish and made some minor changes but can't get it to work".

My advice is to forget Suckerfish or any other 'out-of-the-box' menu or menu generating software. Sooner or later it will not perform as you expected and you won't know why.

Start by understanding the building blocks of creating css driven menus. Then you can put your own vertical or horizontal menu templates together and you will understand how they work. If and when the need arises, you will have a good idea where to start 'tweaking'. I shall assume nothing and take this tutorial from basics, so forgive me if you understand some of this already.

The basic building block of a css menu system is the unordered list as shown in this example:

```
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>...</li>
</ul>
```

Sub-menus are implemented as lists within a list item as shown below:

```
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>
    <ul>
      <li>sub item 1</li>
      <li>sub item 2</li>
      <li>...</li>
    </ul>
  </li>
  <li>...</li>
</ul>
```

This structure is the same whether you are ultimately implementing a vertical or a horizontal menu system.

This system obviously degrades gracefully because it has natural structure and in the absence of any css, the two menu structures above appear as shown below:.

- item 1
- item 2
- ...

- item 1
- item 2
- - sub item 1
 - sub item 2
 - ...
- ...

Taking care of basics

There are three ways to identify a block of styling code. By html element type such as **body**, **div**, **p**, etc. By setting up a class using dot notation, ie, **.myclass**. And by id using the # symbol, ie, **#myid1**, **#myid2**.

As you will see as we start coding, you can string these together so as to be more specific about areas the code will affect.

Our menu structure will be contained within a <div> element as this gives an overall enclosing element that we can do various things with such as position and size as required.

I also recommend that you have your styling code in a file separate from your html.

If you are new to this idea, a stylesheet file has the file type **.css**. There are a number of ways you can call this file from with your html file but I usually stick with:

```
<link rel="StyleSheet" type="text/css" href="mystyle.css" />
```

Before we start styling the menu the way we want, we need to take away some things we don't want. We don't want the browsers default list image and we don't want indentation that can be caused by margins or padding depending on the browser.

Thus are first styling code is as follows:

```
div#navigation {  
  /* Coding as required */  
}  
  
div#navigation ul {  
  list-style: none;  
  list-image: none;  
  margin: 0;  
  padding: 0;  
}
```

Using **div#navigation** in our styling code rather than just **#navigation** avoids a bug in IE which stops sub-menus from dynamically displaying if not used.

list-style: none; gets rid of the discs, circles or whatever your browser's defaults are. This should also get rid of any user defined list images you may have used elsewhere, but guess what, not in IE. Hence the addition of **list-image: none;** just to be on the safe side.

Applying this to the above list structures, they display as:

```
item 1  
item 2  
...  
item 1  
item 2
```

Note that items are no longer indented including the sub-menu items.

```
sub item 1  
sub item 2  
...  
...
```

From this point on, so as to hopefully help clarity, only new or altered styling code will be shown black. Unchanged code will be grey. Also we will only use the html list structure with the sub-menu.

Absolute relatives

The use of absolute and relative positioning of elements with css is something that a lot of people get confused over. Which to use when and there are some who would say never. But my view is that that is an extreme view. Everything has its place when used properly and in moderation.

As far as our menu system is concerned, they have a very real purpose because we need to make use of the way absolute and relative positioned elements react together.

If you have an enclosing element such as a `<div>` element, which has **relative** positioning, then anything contained within it, which has **absolute** positioning, is positioned with respect to its parent, relatively positioned, element. Similarly, a relatively positioned element acts as an anchor point for any absolutely positioned child elements.

You can specify an element as **position: relative** without specifying offsets. It just sets up the structure.

So applying these principles to our menu structure we have the following situation.

```
<div id='navigation'> position: relative;
  <ul> position: absolute;
    <li>...</li> position: relative;
    <li>...</li>
    <li>
      <ul> position: absolute;
        <li>...</li> position: relative;
        <li>...</li>
      </ul>
    </li>
  </ul>
</div>
```

Note that each time the items in the structure move in a level, the positioning type changes. You can of course keep this going to whatever level you need.

And then they were gone

Now, of course, we don't want our sub-menus to show until such time as we roll over their parent menu item. If you are familiar with hiding things with css, you might think you have a choice here between **visibility: hidden** and **display: none** but you don't.

With **visibility: hidden** when the browser is working out how to display the page, it leaves space on the page for the invisible item. With **display: none** no space is created and the page is displayed as if nothing were there. This is the effect we require for our sub-menus.

Implementing this knowledge in our css coding gives us the following:

```
div#navigation {  
  /* Coding as required for page location*/  
  position: relative;  
}
```

```
div#navigation ul {  
  list-style: none;  
  list-image: none;  
  margin: 0;  
  padding: 0;  
  position: absolute;  
}
```

What this coding means is; any element whose parent is a <div> element whose id is **navigation**.

```
div#navigation li {  
  position: relative;  
}
```

```
div#navigation ul ul {  
  position: absolute;  
  display: none;  
}
```

Similarly, this code means any element that resides within a element, which resides within a <div> element that has the id of **navigation**.

And when applied to our menu structure;

```
<div id='navigation'>  
  <ul>  
    <li>item 1</li>  
    <li>item 2</li>  
    <li>submenu  
      <ul>  
        <li>sub item 1</li>  
        <li>sub item 2</li>  
        <li>sub item 3</li>  
      </ul>  
    </li>  
    <li>item 3</li>  
  </ul>  
</div>
```

it now displays like this:

```
item 1  
item 2  
submenu  
item 3
```

Class distinction and id required

Before we go any further, lets put some **class** and **id** attributes in place in our structure as we are going to need them as we progress. The attributes can be seen in the structure below:

```
<div id='navigation'>
  <ul class='level1'>
    <li>item 1</li>
    <li>item 2</li>
    <li class='submenu' >submenu
      <ul class='level2' id='sub1'>
        <li>sub item 1</li>
        <li>sub item 2</li>
        <li>sub item 3</li>
      </ul>
    </li>
    <li>item 3</li>
  </ul>
</div>
```

Note that the name given to a **class** attribute can appear any number of times within a page but the name given to an **id** attribute must only appear once.

This is useful as it allows us to set styling code that is common across a group of elements and also element by element specific styling code.

Our working menu system is not exactly complex but you get the idea.

IE, IE it's going to work, it will

We might as well deal with something that IE doesn't implement properly (well there's a surprise) at this stage, as it's relevant to the proper working of our menu system.

According to the CSS Specification the pseudo-class **:hover** can be applied to any element but IE only applies it to the <a> element. We are going to fix this by using some code that only IE understands to call in a special file of code that makes IE apply the pseudo-class as intended.

This file is called csshover.htc and you can get the latest version of this file and some useful info by going to:

<http://www.xs4all.nl/~peterned/csshover.html>

We call this file using the following code:

```
body {
  behavior: url(csshover.htc);
}
```

Now if you include this in your main stylesheet it will not validate. So in case this is important to you, and it should be, save this code in a separate file. I call mine:

ieonly.css

And then call this file from within your xhtml pages using the following lines of code:

```
<!--[if IE]>
  <link rel="StyleSheet" type="text/css" href="ieonly.css" />
<![endif]-->
```

Note that this 'conditional code', as it is known, is inside comment tags. It is only understood by IE and ignored by other browsers. The code to deal with IE's other little css quirks should also be placed in this file rather than using the many 'hacks' available.

This is the preferred and Microsoft recommended approach and will save you headaches when IE7 hits the tracks.

Rollover and all will be revealed

Add this new block of style code to your stylesheet:

```
div#navigation li.submenu:hover ul.level2{
  display: block;
  width: 150px;
  left: 55px;
  top: 0px;
}
```

You should be getting the hang of this now. This block of code will effect any element that belongs to the **level2** class that has a parent element that belongs to the **submenu** class with the cursor currently hovering over it, and that resides within a <div> element that has the id of **navigation**.

and apply it to our structure from above

```
<div id='navigation'>
  <ul class='level1'>
    <li>item 1</li>
    <li>item 2</li>
    <li class='submenu' >submenu
      <ul class='level2' id='sub1'>
        <li>sub item 1</li>
        <li>sub item 2</li>
        <li>sub item 3</li>
      </ul>
    </li>
    <li>item 3</li>
  </ul>
</div>
```

You still get this:

```
item 1
item 2
submenu
item 3
```

but when you roll the cursor over 'submenu' you get this:

```
item 1
item 2
submenu sub item 1
item 3  sub item 2
        sub item 3
```

Voila! Your first dynamic menu system. Notice also that if you move the cursor (swiftly at least in IE anyway - we'll take care of this need later) onto the sub-menu items the sub-menu remains visible. Only when you move off the sub-menu does it disappear.

Fundamentally, that's it. Irrespective of whether you are constructing a vertical or a horizontal menu, the coding above is what produces a **pure css driven dynamic menu system**.

Everything else is a about appearance and position, which may include one or two browser specific tweaks.