Unobtrusive JavaScript (Sample Chapter)

Written by Christian Heilmann

Version 1.0, July 2005-07-05

This is a free sample chapter, the whole course book is around 60 pages and covers the topic of Unobtrusive JavaScript in-depth with over 50 examples and demonstration files.

You can obtain the complete course by donating a 2 figure amount at http://onlinetools.org/articles/unobtrusivejavascript/

Separation of CSS and JavaScript

You might have noticed in the previous examples that we are not exactly practising what we preach.

Our mistake

True, we separated structure from behaviour, but we also defined presentation via JavaScript, which is just as bad. We defined presentation by accessing the style property of elements:

JavaScript

```
if(!document.getElementById('errormsg'))
{
 var em=document.createElement('p');
 em.id='errormsg';
 em.style.border='2px solid #c00';
 em.style.padding='5px';
 em.style.width='20em';
 em.appendChild(document.createTextNode('Please enter
 or change the fields marked with a '));
 i=document.createElement('img');
 i.src='img/alert.gif';
 i.alt='Error';
 i.title='This field has an error!';
 em.appendChild(i);
}
```

This means that whenever we want to change the look of our enhancement, we'd have to change the JavaScript, and that is neither practical nor clever.

There are several ways to apply presentation defined in the CSS to an element via JavaScript. One would be to apply an ID to the element by changing its ID attribute. A much more versatile way is to apply a class to the element, especially as elements can have more than one class.

Multiple class syntax

For an element to have more than one class, we simply separate the class names with a space: `<div class="special highlight kids">`. This is supported by most modern browsers, some of which show some bugs, though. MSIE on a Mac does not like several classes when one class contains the name of the other, and behaves badly when the class attribute starts or ends with a space.

Applying classes via JavaScript

To add a class to a certain element, we change its `className` attribute. As an example, we could check if the DOM is supported by the current browser and add a class called "`isDOM`" to a navigation with the ID "`nav`". This allows the designer to define two different states of the same navigation in the Style Sheet.

HTML

```
<ul id="nav">
 <!-- more HTML -->
</ul>
```

JavaScript

```
if(document.getElementById && document.createTextNode){return;}
  if(document.getElementById('nav'))
  {
    document.getElementById('nav').className='isDOM';
  }
}
```

CSS

```
ul#nav
{
    /* Non-DOM styles */
}
ul#nav.isDOM
{
    /* DOM enabledstyles */
}
```

However, if we simply change the className attribute, we might overwrite already existing classes. That is why we need to check if there is already a class applied and add the new one preceded by a space.

JavaScript

```
if(document.getElementById && document.createTextNode){return;}
var n=document.getElementById('nav');
if(n)
{
  n.className+=n.className?' isDOM':'isDOM';
}
```

We run into the same issues when want to remove classes that have been added dynamically, as some browsers don't allow for spaces before the first or after the last class. This can be pretty annoying, and it is easier to re-use a function that does this for us (`cssjs.js`).

JavaScript

```
function cssjs(a,o,cl,c2)
{
  switch (a)
  {
    case 'swap':
      o.className=!cssjs('check',o,cl)?o.className.replace(c2,cl):
      o.className.replace(cl,c2);
    break;
    case 'add':
      if(!cssjs('check',o,cl)){o.className+=o.className?' '+cl:cl;}
    break;
    case 'remove':
      var rep=o.className.match(' '+cl)?' '+cl:cl;
      o.className=o.className.replace(rep,'');
    break;
    case 'check':
      return new RegExp('\\b'+cl+'\\b').test(o.className)
    break;
  }
}
```

This function takes four parameters:

- a — defines the action you want the function to perform.
- o — the object in question.
- c1 — the name of the first class
- c2 — the name of the second class

Possible actions are:

- `swap` — replaces class `c1` with class `c2` in object `o`.
- `add` — adds class `c1` to the object `o`.
- `remove` — removes class `c1` from the object `o`.
- `check` — tests if class `c1` is already applied to object `o` and returns `true` or `false`.

Let's create an example how to use it. We want all headlines of the second level expand and collapse their next sibling element. A class should be applied to the headline to indicate that it is a dynamic element and to the next sibling to hide it. Once the headline gets activated, it should get another style and the sibling element should get shown. To make things more interesting, we also want a rollover effect on the headline.

JavaScript

```
function collapse()
{
  // check for the DOM and initialise variables
  if(!document.getElementById || !document.createTextNode){return;}
  var heads,i,p,tohide,activeMessage;
  /* CSS Classes */
  var hc='hidden';   // hide an element
  var sc='shown';   // show an element
  var tc='trigger';   // indicate the element hides and shows another
  var oc='open';     // indicate an trigger preceding an open section
  var hoc='hover';   // hover style of the trigger element
  /* CSS Classes end */

  // create a new paragraph with the message that the headlines are
  // interactive
  p=document.createElement('p');
  activeMessage='Click the headlines to collapse and expand';
  activeMessage+=' the following section';
  p.appendChild(document.createTextNode(activeMessage));
  // loop through all headers of the second level
  heads=document.getElementsByTagName('h2');
  for(i=0;i<heads.length;i++)
  {
    // find the next sibling and hide it with the appropriate class
    var tohide=heads[i].nextSibling;
    while(tohide.nodeType!=1)
    {
      tohide=tohide.nextSibling;
    }
    cssjs('add',tohide,hc)
    // add the trigger class to indicate that the header is clickable
    cssjs('add',heads[i],tc)
    heads[i].tohide=tohide;
    heads[i].tohide=tohide;
    // add and remove the hover class when the mouse moves in or out
    heads[i].onmouseover=function()
    {
      cssjs('add',this,hoc);
    }
    heads[i].onmouseout=function()
    {
      cssjs('remove',this,hoc);
    }
```

JavaScript

```
..continued
    // swap the open and trigger and show and hide classes when
    // the headline is clicked
    heads[i].onclick=function()
    {
      if(cssjs('check',this.tohide,hc))
      {
        cssjs('swap',this,tc,oc);
        cssjs('swap',this.tohide,hc,sc);
      } else {
        cssjs('swap',this,oc,tc);
        cssjs('swap',this.tohide,sc,hc);
      }
    }
    // insert the message that the headers are interactive before
    // the first header
    document.body.insertBefore(p,document.getElementsByTagName('h2')[0]);
  }
  function cssjs(a,o,c1,c2)
  {
    // ...
  }
}
window.onload=collapse;
```

CSS

```
.hidden{
  display:none;
}
.shown{
  display:block;
}
.trigger{
  background:#ccf;
}
.open{
  background:#66f;
}
.hover{
  background:#99c;
}
```

This example is contained in the accompanying zip file as ex_cssseparation.html.

We now have successfully separated structure, presentation and behaviour and created a rather complex effect. Maintenance of this effect is easy, and does not require any JavaScript knowledge.