

/CSS/sort out your drawers



Craig Grannell provides you with a quick, easy and modular solution for creating website drawers, using a lean little script that's just hundreds of bytes in size

Knowledge needed Intermediate CSS, JavaScript and HTML

Requires A text editor

Project time 15 minutes

Sometimes, there's not enough room onscreen for all the information you want to provide, or your web page contains supplementary information that you don't want to display because it distracts users from the main content. In cases like these, drawers have become a popular way of adding to your page's effective real estate, enabling you to hide content behind a clickable component, only displaying the content when users request it.

Various methods exist for creating drawers, including solutions built into popular applications, components of libraries such as Scriptaculous, and freely available JavaScript solutions on the web. The problem with application solutions and libraries is that the code can be weighty. If you just want some drawers, there's no point in forcing users to download large library files. As for web solutions, most of them work on the basis of single targets. In other words, the trigger toggles the display property of an element with a specific id. That's fine if you only want a single drawer, but when you have plenty, a system like this can be tedious to work with.

In this tutorial, you'll find out how to use an efficient piece of JavaScript to create drawers in a modular fashion. As long as you use the same basic structure for each drawer, the set-up remains identical throughout, saving you having to faff about with id values!

A basic example

The 'basic-drawer' folder on the CD includes a simple example. Open 'collapsible-div-modular.html' in a browser and click the **Toggle drawer!** link twice. You'll see previously hidden content display and disappear. The web page is simple: in the **head** section, the CSS document 'drawers.css' is imported, and two JavaScript documents are linked to, namely 'collapsible-

div-modular.js' and 'javascript-overrides.js'. In the page body is the code that drives the drawer. In abbreviated form, it looks like this:

```
<div class="container">
  <h2><a href="#" title="Toggle drawer" onclick="toggle(this); return
false;">Toggle drawer!</a></h2>
  <div class="expandable">
    <p>[...]</p>
  </div>
</div>
```

The **container** div is primarily a CSS hook for constraining the drawer's width and adding a border. The level-two heading's content is within an anchor, whose **onclick** event triggers the **toggle** function. The **this** value applies the function to the anchor (rather than an element with a specific id value, which is what most scripts do). Following the heading is a div with a **class** value of **expandable**, inside which is the initially hidden content.

The CSS in 'drawers.css' is simple. The only thing to note is that the dotted border separating the heading and **expandable** div (when the hidden content is shown) is applied to the top of the **expandable** div, not the bottom of the heading. This ensures that it's only shown when the **expandable** div is visible.

The JavaScript document 'collapsible-div-modular.js' houses the **toggle** function. The initial line of the script traverses the DOM tree, from the anchor element that's been clicked, to its parent (the level-two heading), and then to the next sibling (the **expandable** div). For browsers that consider the white space between the level-two heading and the div to be content, a fallback is included, moving the target on from the white space to the **expandable** div.

```
if(document.getElementById) {
  targetElement = toggler.parentNode.nextSibling;
```

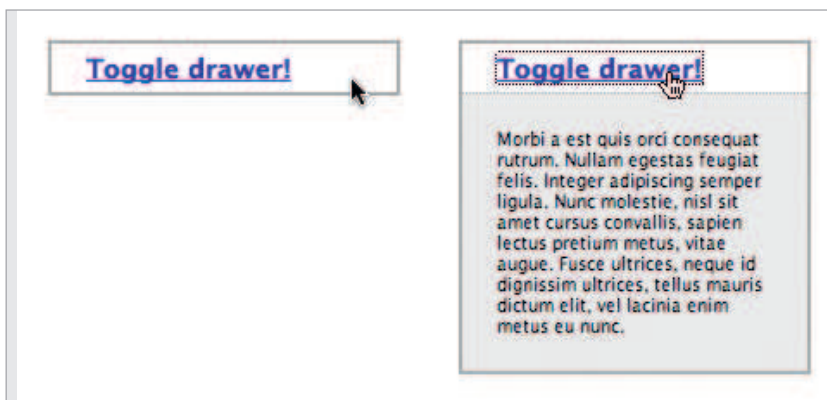
```
if(targetElement.className == undefined) {
  targetElement = toggler.parentNode.nextSibling.nextSibling;
```

The remaining lines in the script determine whether the **display** property of the target element is set to **block**. If it is, the property is set to **none**. If not, it's set to **block**.

Ensuring accessibility

A common problem with scripts like this occurs in making initially hidden content invisible by setting its container's **display** property to **none** in CSS (which we could have done via the **.expandable** rule). This means that users accessing the page with JavaScript disabled have no means of accessing the hidden content, since the **display** property is toggled using JavaScript. Instead, the **none** value should be defined using JavaScript.

This can be done in various ways, but to keep things modular and extensible, it's best to create a separate style sheet for JavaScript overrides and attach it using JavaScript. Therefore, in the 'basic-drawer' folder, you'll see 'javascript-overrides.css' (with a single rule, setting **display: none** for **.expandable**) and 'javascript-overrides.js'. The JavaScript document writes the relevant markup to the web page to link the overrides CSS document. For



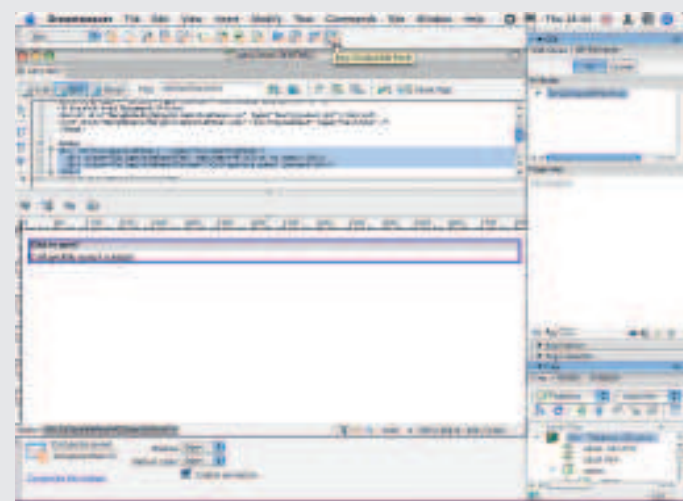
On the disc This tutorial's 'toggable' drawer is suitable for various site functions: a gallery thumbnails drawer, navigation, or a question-and-answer drawer in an FAQ page

Cleaning up with Ajax

Fancy taking the automated route instead?

Elsewhere in this tutorial, I dismiss Ajax for drawer functionality on the basis of page weight. However, if your site is likely to make use of other components of a framework or library – or if you’ve got a shiny new copy of Dreamweaver CS3 that you’re desperate to justify on the basis of its new features – the automated route can be quick and easy. Scriptaculous makes it possible to add drawers with animation simply by using `Effect.toggle` and then defining an effect type (such as `blind`) and a duration. And, as mentioned earlier, Dreamweaver now has some Ajax components built in, by way of Adobe’s Spry framework, which is great for visual designers who hate getting their hands dirty with code.

However, take care when using off-the-shelf solutions. In the short term, they offer speed benefits, but they may bloat your site. Also, the Scriptaculous example relies on targeting elements with a specific id value, which is good for a one-off, or for targeting an element elsewhere on the page, but tedious to set up if you have a number of drawers.



Ready-made solutions Dreamweaver 3 comes with built-in, easy-to-use Ajax components, though it may still be more efficient to use your own equivalents



Saving space Because Images from Iceland (www.snubcommunications.com/iceland/iceland-old.html) uses drawers in its interface, it has many thumbnails on a single page

users with JavaScript enabled, the overrides CSS is loaded (so content intended to be initially hidden will not be visible when the page loads). For users with JavaScript disabled, the markup necessary to access 'javascript-overrides.css' isn't written to the page, and so the content intended to be hidden is shown, which is generally better than it being inaccessible. (Note: this may not be the case for all websites, so use your judgement when making such decisions.)

The code for writing the markup is shown below. You might be used to seeing `document.write` for this kind of thing, but it isn't valid with XHTML (see 'Resources', page 72), hence building the `link` element using the code shown.

```
var cssNode = document.createElement('link');
cssNode.setAttribute('rel', 'stylesheet');
cssNode.setAttribute('type', 'text/css');
cssNode.setAttribute('href', 'javascript-overrides.css');
document.getElementsByTagName('head')[0].appendChild(cssNode);
```

Expanding the concept

With a modular method of creating drawers, it's easier to get creative. In the 'horizontal-drawer' folder on the CD, there's a more advanced version of the drawers concept. Open 'drawers.html' in a browser and you'll see three vertical strips with icons at the top of each one. Clicking the icons toggles the drawers, which open horizontally. Despite this design's major presentation differences to the basic example already discussed, the markup is similar. In terms of changes, in the `head` section there are two conditional comments to deal with Internet Explorer shortcomings. In the body section, the page's content is within a `wrapper` div, and there are three drawers instead of one. Each of the `container` divs now has a second `class` value, used as a CSS hook, providing individual styles for each `container` div.

When it comes to the content of the divs, there are few noticeable changes. The main one is in the headings, the content of each one now being a clickable image and an explanatory piece of text shown when the drawer is expanded. Elsewhere, the only other item of note is the `separator clearing` div, prior to the `wrapper` div's end tag. Because the drawers are floated using CSS, the background applied to the `wrapper` div in CSS (a PNG gradient, horizontally tiled) won't show if this content isn't cleared. While Position Is Everything's `clearfix` technique (again, see 'Resources', page 72) would ideally

be used in this situation, it doesn't work in Internet Explorer for this design, hence dragging the `clearing` div method out of retirement.

In 'drawers.css', you'll find the styles for the expanded page. The first three rules reset margins, padding and font-size values, with the `body` rule adding a tiled background GIF. Next, the `#wrapper` rule sets padding around the wrapper's content, sets the PNG gradient as a horizontally tiling background, and defines a default font and colour. Note: the PNG graphic isn't eye candy – the darker top for the page draws the eye to the icons. Next, the `.separator` rule styles the `clearing` div, ensuring the wrapper's background displays as intended in all browsers.

Styling the drawers

Because of the nature of this design, some trickery is required in CSS. The headings contain a linked icon for toggling the drawer and the content displayed when the drawer is expanded. Therefore, the container's `overflow` property must be set to `hidden`, which crops content when a drawer is closed. This is defined in the `.container` rule (see the following code block), which also floats the containers, defines their height (in ems, since that expands with text zooming, unlike a pixel-based setting) and a width value (which is the value of the expanded drawer – the default narrow width setting is defined via the override JavaScript, otherwise those with JavaScript disabled would never see the content within the drawers).

```
.container {
float: left;
height: 45em;
}
```

The headings contain a linked icon for toggling the drawer and the content displayed



Quick test When working with websites that have JavaScript functionality, it's a good idea to use Opera's **Quick Preferences** to test how things work if JavaScript is disabled

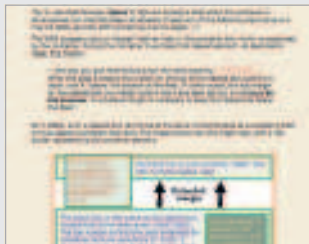
```
>> width: 250px;
    overflow: hidden;
}
```

The next three rules, `.photography`, `.music` and `.gaming`, define background colours for the three individual drawers, while `.container h2 img` floats the images within the level-two headings left and defines a right-hand margin, so subsequent content (text within the headings) doesn't abut the images.

The `.container h2` rule defines the full width of the heading (for IE!), sets `white-space` to `nowrap` (so the heading stays on a single line, rather than line-breaking when its container's width is reduced), and defines styles for the text content. The `.expandable` rule then defines settings for the `expandable` div, setting a top border, a left-margin (defined so the underline begins at the same horizontal position as each heading's text content) and padding.

```
.container h2 {
width: 250px;
white-space: nowrap;
text-transform: uppercase;
font-size: 2em;
line-height: 2em;
}
.expandable {
```

Resources Find out more online



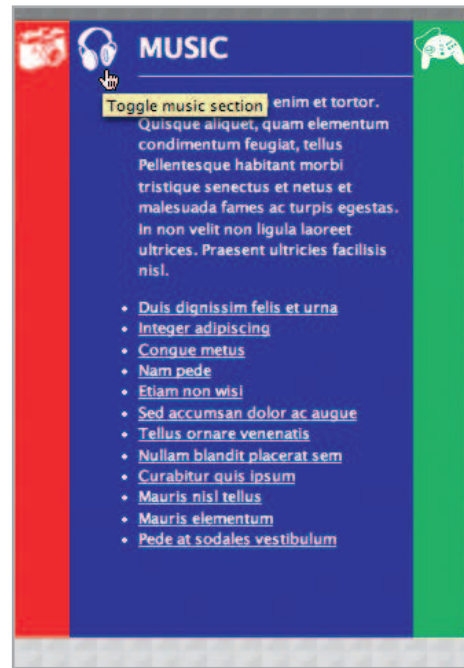
Position Is Everything
I've used a clearing div, as per W3C's guidelines for clearing floated content. The **clearfix** method on Position Is Everything is better for general use.
www.positioniseverything.net/easyclearing.html



W3C FAQ
This page outlines the differences between XHTML and HTML. Point 14 provides confirmation from W3C about the invalid nature of `document.write` in XHTML.
www.w3.org/MarkUp/2004/xhtml-faq.html



My example page
The second togglable div is being clicked, opening the **music** section. This example shows the flexibility of the system: the drawers open horizontally, not vertically, as in my basic example



```
border-top: 1px solid #ffffff;
margin-left: 50px;
padding: 10px 10px 10px 0;
}
```

The remainder of the rules style the page's other text elements and links, and remove the border from images within links. The JavaScript override for CSS works as per the basic example, except there's an additional rule to set the default width of the containers. See the `.container` rule in 'javascript-overrides.css', where `width` is set to 40px.

Updates and fixes

The 'drawers.js' document is different to the basic example, because the toggle function now dynamically alters two CSS properties: the `display` property of the relevant expandable div (as before) and the `width` setting of the parent container div. This is why I created the new variable `targetDiv` (whose value is the parent of the parent of the relevant anchor element, and always a div with a `class` of `container`), and the `targetDiv.style.width` lines in the `if/else` portion of the function. There, the widths toggle between the default width of 40px for the container divs when the expanded drawer isn't visible, and the expanded width of 250px when the drawer is visible.

Finally, IE fixes are required, applied via style sheets attached using conditional comments. The first is for IE7, which doesn't correctly show the transparency of the wrapper div's PNG background. This is likely to be a `hasLayout` issue, because by giving the wrapper div layout – using the Holly hack to set the `height` of `#wrapper` to 1% – the problem goes away (see 'ie-7-hacks.css'). Previous versions of IE can't deal with PNG transparency at all, but this can be fixed with a proprietary IE property, or by setting `background` to `none` for `#wrapper` in an IE6 and lower CSS document (as in the example files).

Of course, this version of the drawers concept is more specific, tying itself to two width values for the drawers. However, you can use a generic and a specific version of the toggle script without compromising code and download times. Just remember to give the functions individual names! ●



About the author

Name | Craig Grannell
Site | www.snubcommunications.com
Areas of expertise | Information architecture, site concepts, graphics, interface and front-end design
Clients | Swim~, Rebellion, IDG
Favourite holiday food | Decent, authentic Mexican