CS 193c Final Project writeup
Peter Solderitsch

Being familiar with many aspects of client-side web design and implementation in my "previous life" working for an internet/"e-business" consultancy, I wanted to focus my project specifically on more advanced and underused features of Javascript and dynamic content and positioning, to better learn things that I didn't know and didn't get a chance to use in the day-to-day process of building commercial, public sites that not only had to display correctly on the latest browser versions with fast hardware, but on old Pentium I pc's with 56k modems (if even that fast).

Obviously, this decision had some drawbacks. One of the reasons that these features *are* underused is because any DHTML application of non-trivial size quickly grows into a big pain to code. If you're going to write an outlining tool, which is what my project is, you're not likely to choose xhtml, css, and javascript as your medium of choice, because doing *everything* requires jumping through hoops, including browser compatibility issues (due to varying degrees of implementation of the w3c specs, or bugs in those implementations) and the nature of the Javascript DOM API, which makes many algorithmically trivial operations a challenge. To cut down on at least some of these pitfalls, I decided very early on in the development process to focus specifically on Mozilla compatibility, sacrificing IE (both on Mac and Windows), for the guarantee that it will work on any platform supporting Mozilla, which includes Mac, Windows, Linux, and more. Even still, *so much time* was spent just polishing up aspects of the interface, making things work smoothly when you click, and trying to minimize javascript/DOM idiosyncracies and annoyances. Overall, I'd give this project the codename "frustration". Hopefully it'll be clear from the source how much it actually took for me to produce even the (arguably) minimal results that you see on screen.

My DHTML mini-application is called SiCSO, for Simple Client-Side Outliner. Essentially it is a clone of many of the basic features *and behaviors* of a program I like for Mac OS X called OmniOutliner (more info on that program can be found at www.omnigroup.com). You can add and edit single lines of text, and arrange those lines into a hierarchy, and up and down within levels. You can adjust the font face and font size of the outline as a whole, and can generate a pop-up window that presents the outline plainly, so it's suitable for printing. Be sure to look for these neat little features (each of which took some time and thought);

- Clicking on any outline text automatically switches the text to an input box for editing. No need for the two-step process of clicking on a form button. The input box also dynamically shrinks/grows depending on how much text there is on the line. Hitting the "return/enter" key when you're finished entering text will convert the text input box back into regular text.

- If a line is currently highlighted, you can indent it by pressing “i”. You can outdent by pressing “I” (that’s ‘Shift-I’) Or, you can use the red toolbar indent/outdent buttons, obviously.
- The “pop-down” instructions pane in the upper left. Mozilla allows you to use alpha-channel PNG images as backgrounds, which provides for cool transparency effects. If you drag on the “instructions” titlebar, you can reposition that “pop-down” window anywhere onscreen. There’s a small bug with that when you first start dragging the first time, but other than that it works. This isn’t integral to the application, it’s just something I wanted to try.
- The “print window” is a cool little hack. First it pops up a popup window with an html file that is basically empty, except for a stylesheet link. Then, it clones the outline nodes from the main window and inserts them into the body of the popup window’s DOM. You’ll notice that it says “Please wait…” when the window first opens. Originally, Mozilla was inserting the cloned nodes *before* the html for the popup was finished loading. So I put in a Javascript delay of 2 seconds before copying the cloned nodes over. Assuming you have a reasonably fast connection, 2 seconds should be enough time. Let me know if it doesn’t work for you, and I can adjust the delay time!

Third-party code:
I used (or, more correctly *adapted*) several bits of third-party code found on the web. These bits were all of the “javascript tutorial” variety and thus I assume to be immune from copyright issues. All of the code I borrowed is explicitly documented (including URLs, where available) in-line as javascript comments, the bulk of which is located in the file “helpers.js” which is in the same subdirectory as the project. The code relates specifically to capturing key press events and drag and drop/mouseMove events.I also repurposed some of the example “hierarchy tree” code presented in the class handouts.

Third-party Images:
The only images I used were “plus” and “minus” images which I fear are probably copyrighted, as I believe that they originated as screen grabs from Windows File Explorer (or similar). I’ve had these on my computer for awhile, and cannot recall/retrace where I originally obtained those graphics.

Code locations:
The actual webpage is at http://www.stanford.edu/~petey/cs193c/project/project.html. The printable template is at http://www.stanford.edu/~petey/cs193c/project/outlinePrint.html.

Javascript is found both in the project.html page directly, and in a file called helpers.js

Styles are found at outline.css and outlinePrint.css.

Possible Extensions:

I didn't get a chance to implement *many* of the features I'd have liked to. Here are just a few things I'd like to do (or have done):

- Multiple simultaneous selections (i.e., a "shift-click"-type behavior to select more than one line at a time)
- Font & size control on a line-by-line basis instead of globally for the whole outline. I had this partially working with font support, but wrangling through the DOM introduced too many bugs, so I had to back it out.
    - Also, 'bold', 'italic', 'underline' support for same.
- Support for auto-numbering schemes (e.g. 1, 1.1, 1.1.1, or I. A. i. a)
- Support for saving, writing to a file, reading data from a file, perhaps in an XML format (OPML is an existing standard XML file format for outline documents, I investigated supporting this but didn't have time to include it).
- Alternate style sheet themes for the outline editor itself.
- Specific support for multi-line outline entries, i.e. using a "textarea" instead of input type "text".
- Support for including html in the entries.
    - Once you had that, it would be neat to create a version of the "file-writing" extension described above that would auto-generate dhtml hierarchical menu code. That is, you could create your own menu graphically via my SiCSO tool, and then it would save the menu as an html file, complete with all the necessary javascript, etc. That would be really cool.