

Developing Ajax Applications

Version 1.0

Instructor's Guide

Overview

This course introduces Ajax programming to an intended audience of software developers who've yet to work much on the client side of web applications, and so are largely unfamiliar with the browser programming model and Ajax in particular. It gives an overview/introduction, and then spends about half of its time on the browser programming model of JavaScript, DOM, and CSS, before digging into Ajax proper by way of XHR. Then, having understood nuts-and-bolts XHR usage, and seen the difficulties involved with that, students get a look at two popular Ajax and JavaScript frameworks: Prototype and Dojo.

Timeline

The following breakdowns are approximate, and every class will vary.

Day 1

2½ hours Chapter 1

4 hours Chapter 2

Day 2

2 hours Chapter 3

2½ hours Chapter 4

2 hours Chapter 5

Day 3

2½ hours Chapter 6

3½ hours Chapter 7

A two-day timeline is possible for those already experienced with JavaScript, DOM, and CSS, where most or all of Chapters 2-4 is skipped or skimmed.

Tools Deployed with the Lab Software

This course's software requires separate setups of a Java DK, the Crimson text editor, Eclipse WTP 2.0, Firefox, and Firebug. The lab installer sets up other necessary tools under **c:\Capstone\Tools**:

- **Ant1.6** – the Ant make utility, which we use to build and deploy web applications
- **Dojo1.1** – the Dojo toolkit
- **JSTL1.2** – the JSP standard tag library, which is deployed with some of our web applications so that they can do some basic on-page processing on the server side
- **Tomcat6.0** – the Tomcat web server and container

Note that none of the separately-installed tools is mentioned prominently in the coursebook. Many are optional, and even the Java DK we downplay, since it's just an engine for the server side in a course that's supposed to be about the client side (and is not Java-specific). Use Eclipse or Crimson at your option – and see the later section on the Eclipse overlays. Firefox is the standard browser for all the exercises. And you may want to introduce Firebug early in the course as a nice diagnostic and debugging tool, and give a quick demonstration of that.

Ant Build Process

Many of the exercises in this course can be tested in place, simply by opening an HTML page in Firefox. Then, some are web applications and need to be deployed to Tomcat. For that purpose we use Ant.

Though most students will be happy to let the **ant** command take care of things, some students (and most instructors) will want to understand the inner workings here a little better. Each project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant** which is typically **c:\Capstone\Ajax\Ant**. Information here defines a routine for building a web application, and deploys a Tomcat context file to let the server know how to map a context root to the **build** tree created by the build.

Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse WTP 2.0 for this course. (See the course Setup Guide for download URLs.) Instructors, use this package on your own initiative and at your own risk. You should have experience yourself with Eclipse before using the overlay package in the classroom. The workspace and projects have been tested lightly with the course but are not part of the standard product.

That said, this overlay should save a good deal of work for those who wish to use Eclipse instead of the text editor and command-line tools that are standard for the course. See the file `c:\Capstone\Ajax\Eclipse\ReadMe.html` for general notes on how to use the Eclipse overlays for Capstone courses. Be prepared to walk students through the first few exercises in Eclipse; the notes in this file are for experienced Eclipse users, and will not be clear to many students on their own.

Be sure to follow the instructions in that ReadMe file that point you to module-specific notes in `c:\Capstone\Ajax\Eclipse\Ajax Module Notes.html`. There are a few key instructions there, without which a handful of the projects (mostly the Dojo ones) will not function in Eclipse.

Again, Capstone Courseware can only offer complete technical support on the standard course, and while we hope this overlay is convenient, it is not as thoroughly tested as the core lab image at this time. If a given exercise is giving trouble, please be certain to build and run it from the command line, using the SDK tools as prescribed in the student guide, before contacting Capstone.

Internet Explorer Issues

Note that course exercises have been tested on Firefox 3.0, and that this browser is required as part of the classroom setup. You may find cross-testing on IE to be illustrative of the sort of issues one encounters in trying to get consistent results across multiple browsers. For example we know of at least these issues on IE6:

- The tab image doesn't float correctly in the **Positioning** example.
- The **Maps** example is a train wreck: the image size doesn't take, the drag-and-drop doesn't work, etc.
- The **Walker** example works, but notice that the text output is different, showing a different DOM tree than that represented in Firefox. This hints at possible problems with DOM programming between the two browsers.
- Some whitespace presentation is different in the `examples.js/examples.css` framework used in many early examples.

Teaching Notes

Chapter 1

In this overview chapter, we try to develop a coherent meaning for the sometimes fuzzy term “Ajax,” and a good way to do that is to describe what Ajax is not. It’s not the traditional browser-hosted application, and the comparisons between them help not only to define Ajax but also to establish that neither the traditional page-per-user-action model nor a pure-Ajax model using a single HTML page and a ton of JavaScript is likely to be the perfect architecture for the typical business application.

The only examples in this chapter are two implementations of the Flights application – one “traditional” and one using Ajax. This should make the basic focus of the course pretty clear: how do we get remote requests moving, and how do we process the responses? Though we do explore some rich-UI features, especially as we get to Prototype and Dojo, we do mostly concern ourselves with the simpler scenarios of remote requests and DOM/CSS updates in response to user actions.

Chapter 2

Our aim in this chapter is to develop enough facility with JavaScript to implement simple Ajax applications using DOM, CSS, and XHR. Without knowing how much JavaScript the students may already know, we have to start from the beginning. So this chapter is rather long, and much of it may be review for your group. Have to play this one as it lies; you may even find that you can skip the chapter entirely – though be aware that doing so will have more than a half-day impact on your total timeline.

For most of this chapter we present JavaScript as a pure programming language, pulled away from the browser context except that the browser is a nice host for JavaScript programs. Most of the exercises are traditional console applications, with minimal user input and plain-text output. The design of the example pages should make it easy to illustrate each programming technique without flipping too much between the presentation PDF, the source file, and the browser.

In most cases things will hum along at least as far as the coverage of functions. When we hit the objects section, things may get complicated. Some students won't be familiar with OO, and that may take some explaining – although rigorous OO theory is hardly necessary for this course. The part that may be surprising is how experienced OO programmers handle this material. It's just familiar enough to be deceiving, and a lot of Java or C# programmers will jump to conclusions about object types and how they work. The general misconception that JavaScript is sort of a Java-light will pretty well explode when you hit the prototypes section – don't be surprised if that's the moment that a bunch of people suddenly realize that they're not getting it, and need to go back a bit.

In past offerings instructors have found that the **examples.js** and **examples.css** files that provide a standard framework for many of the code examples tend to draw students' attention. They're not intended to, or at least not this early in the course, as they use some techniques we don't cover for a while yet. But you may find that students are curious about why the page looks/works the way it does. It's certainly fine to delve into these files whenever. Do beware of them as a possible time drain though.

Chapter 3

The DOM is another area in which students may or may not have prior experience – but for a different reason. Even if one has not coded to the HTML DOM using JavaScript, any experience with the XML DOM on the server side will translate very well. So much of this chapter may feel familiar to some, and perhaps only the global objects such as **document** and **window**, the event model, and the **innerHTML** usage will provide new information. Again, it will be important to sound out the audience ahead of time, and proceed accordingly.

Chapter 4

This chapter offers some surprises for most developers – even if they have prior CSS experience. Most who’ve worked with CSS know it as a way of getting the right font, color, and text-styling choices in place, but have not worked so much with the layout model. And of course this is a potent piece of modern CSS. Dynamic styling such as rollovers and show/hide responses should be eye-openers.

But the really winning part of this chapter is the maps case study at the end. At this point in the course we’re in a position to put it all together and start doing some pretty compelling things with the HTML user interface, and this series of demo, lab, and final demo offers a nice chewy exercise that caps off the three-chapter unit on browser programming. (Beware, though, that as we get more adventurous with our CSS, some browsers will not support the code. IE6 is pretty well lost right away with this example, and the default browser in Eclipse won’t do any better. Use Firefox, and configure Eclipse to use Firefox, at least for this part of the course. (Most everything else does work in IE without any trouble.)

One almost scary quirk of the Maps example is worth mentioning. Try removing the whitespace between the start and end tags of the inner div, so it’s `<div id="mapImage"></div>`. Test, and you’ll see the dragging behavior all gummed up in Firefox. It seems there’s a default behavior of letting the user drag image URLs off the page. For an `` tag, this is defeated by writing one’s own **onmousedown** handler – even a dummy one, as shown in the final step for the pushpin image. But for a background image, apparently, the behavior kicks in only if the element has no foreground content. So, even a single space will make this go away – but with no whitespace it’s a huge mess. To my knowledge this is completely undocumented (and, not to put too fine a point on it, was a killer to track down).

Chapter 5

Now we add remoting to our toolbox. We can fire a request – and when we get a response we have the skills from the previous three chapters to express that response to the user whatever way we want. The chapter has two main sections: one on how to make requests and responses move around – considering iFrames first, and then XHR, and then noting some technical issues in avoiding race conditions – and then working with the most common Ajax data formats. The two examples in this second section are, as much as possible, an apples-to-apples comparison of XML and JSON usage.

Chapter 6

With this chapter we move beyond the minimal environment of JavaScript, DOM, CSS, and XHR – techniques available with support from nothing but the chosen web browser – and start to consider frameworks and toolkits. In each of this and the next chapter we introduce some new UI features and programming practices – in no particular order, and not trying for completeness with either Prototype or Dojo, but just giving a sampling of what higher-level browser programming can be. Of course there's a focus on Ajax support, and so we give Ajax.Request a good airing, as we do with dojo.xhrGet/Post later.

The other highlight of this chapter is the bit on algorithmic programming with Prototype's enhanced **Array** and **Element** types. This is powerful stuff, taking the best possible advantage of one of JavaScript's strong suits – closures – and facilitating a programming style that's highly productive and actually pretty hard to do in most server-side languages. The lab for this chapter folds this new technique into the Flights application, and this should make a strong impression.

Chapter 7

We close with Dojo, and go deeper with UI features along with basic remote-request support. There's a little more to work with in Dojo, for one thing, but it's also a nice way to close on some high notes, as the widget system is impressive – especially the library of built-in components.

Dojo is also heavier to work with than Prototype, and students will see and feel that in the exercises. Pages take longer to load, web applications take a lot longer to deploy, and you'll probably observe a new phenomenon, which is an application not actually being available for quite a while after Tomcat has begun its deployment process. That is, you'll see a 404 page if you try to visit the newly-deployed Dojo application too quickly. You might show this to students early, and condition them to wait 10 or 20 seconds after seeing that Tomcat's noticed the new application before testing.

There are also some glitches in the Dojo widget system that make the code less than perfectly portable from one web context root to another. That is, relocating an application by changing the context root requires source-code changes. This is no problem when using Ant, but it gives Eclipse fits. If you're using Eclipse in your class, see the module notes for the Eclipse overlay and get familiar with the workarounds that are needed to get this chapter's exercises working in Eclipse.

Revision History

Version 1.0 is the initial release of the course.

Troubleshooting

If you run into any trouble with code exercises, the first and best thing to do is to double-check that the classroom machines have been set up precisely according to the course setup guide. Especially, the wrong version of a tool can cause significant problems; don't wander off-book in this way unless absolutely sure you can support the software that you prefer and that we haven't tested. Check environment variable settings carefully, too; these are the cause of a great many classroom glitches.

Below are some specific pitfalls that have come up in previous offerings of the course:

- Firebug is the best diagnostic and debugging tool at your disposal in the standard classroom setup. Enable and open Firebug and test a page that's giving you trouble. It can show all the HTTP requests made by the browser (which sometimes highlights a mistyped path or something like that), trace JavaScript, explore the resident DOM tree, and much more.
- In the Chapter-5 example on XHR and race conditions: be careful to wait a few seconds before clicking Clear and re-testing. Remember that a response may still be returning to the browser! So the scenario to avoid, and which makes it seem like the example code is broken, is to click one or both of the request buttons, then Clear, in quick succession: the results fields will clear, but then one may suddenly show results, even before you appear to have requested any.
- If students run into any trouble in Lab 7B, try adding **isDebug: true**, to the **djConfig** attribute in **showMemo.html**. This will float a debug console over the page, and specific error messages and useful information can sometimes be found there.

Errata

At this time there are no errata for the course.

Feedback

We very much appreciate whatever feedback we can get on our courseware – especially from the instructor’s perspective. Naturally, the more specific, the better, and we strongly encourage you to make notes on issues you may encounter in the classroom, whether they’re typos, missing files, or suggestions for clearer language to explain a concept. We can’t guarantee that we’ll act on every suggestion, but we’re aggressive about stamping out problems and try to be highly responsive. Hopefully this means that when you give us good feedback, you get a better course the next time you need to teach it.

Please direct all courseware feedback to

Will Provost
Capstone Courseware
<mailto:provost@capcourse.com>
877-227-2477

For anyone who’s interested, we have a very informal defect-tracking system, based in Excel spreadsheets with columns to capture defect location, nature, status, and author feedback. Ultimately, feedback goes into these sheets, so if you want a template, we’ll be happy to provide one, to facilitate the reporting process.