CHAPTER 12

Ajax: Where the Client and Server Collide

Ajax is all the rage today. It seems that you can't even be a web developer these days without knowing at least something about Ajax!

In this chapter, you will learn a bit about Ajax and put it to use building a one-on-one support chat application, as you can see at the sites of many companies that provide live chat support for their products and services. This will be the one project in this book that uses a server component as well, so you'll see how that all fits together. Additionally, you'll be introduced to a new library, Mootools, and see what it has to offer.

Chat System Requirements and Goals

Let's make believe for a bit. Say we're a new company on the block. We'll call ourselves Metacusoft Systems, for no other reason than the domain name is available and Googling for it returns no results, so I can be reasonably sure I'm not infringing on anyone. The first reader to register the domain name and start a company gets a prize! (No, you really don't, but feel free to take the name.)

Anyway, we sell widgets. Yes, the typical, mundane widget product. It's wonderful, it's amazing, and no one can live without it, and yet we can't come up with a more descriptive name. And, as great a product as it obviously is, it's not always a completely smooth ride for our customers. Sometimes, the widget doesn't, um, widgetate, as it should. Sometimes, although we would never admit this in a financial filing, customers need some assistance to effectively change their lives for the better with our amazing widgets. So, we need to offer some support for them.

We'll have a couple of people available via phone (over in India, of course, where there's apparently no shortage of folks named Bill, Mike, Tom, Sam, and Dave). We'll also offer something for the online crowd, those people who, like the Morlocks,[1] shun human contact. We'll offer a live one-on-one chat system where customers can communicate with human beings in real time, without needing to really talk to them and risk an actual conversation.

1. Morlock is the name of an invented species, offshoots of the human race, created by the famous author H.G. Wells in the novel *The Time Machine*, who exist many, many, *many* years in the future (the Morlocks, not H.G. Wells). The Morlocks live underground and are at that point almost not even identifiable as humans (formerly so anyway). Oh yeah, the Morlocks eat a species known as the Eloi, who are also descendants of the human race. Nice, huh?

It's a relatively simple application—one person types, the other sees it, and that's about the extent of it. Still, we'll try to jazz it up just a bit.

- The application should look halfway decent. After all, this is the public face of our company for those having difficulties with our product. They're probably already a little ticked at us, and we don't want to annoy them further with an ugly web site!
- Let's allow customers the ability to copy a transcript of their chat to their clipboard for pasting in another application. This way, they can have a record of their conversation in case they need to call and yell at us later for something. Let's also give them the ability to print the transcript directly from the application, so that when they sue, they have the documentation they need to win (perhaps we're being a little too good to our customers?).
- The application should of course be Ajax-based and should use the Mootools library to provide that functionality (along with anything else we may need that it offers). It's pretty tough to do a one-on-one chat in HTML and JavaScript without a server component, so we'll need a server in the mix here. We'll code the back-end in Java and Microsoft technologies (ASP specifically), so that at least a majority of developers reading this should be covered.

With those really pretty limited set of requirements in tow, let's get a move on and see how we'll put this thing together, shall we? Well, in fact, let's first look at how we *won't* be doing things and why.

The "Classic" Web Model

In the beginning, there was the Web. And it was good. All manner of catchy new words, phrases, and terms entered the lexicon, and we felt all the more cooler saying them. (Come on, admit it, you felt like Spock the first couple of times you used the word "hypertext" in conversation, didn't you?) *Webapps*, as our work came to be known, were born. These applications were, in a sense, a throwback to years gone by when applications were hosted on "big iron" and were accessed in a time-share fashion, since no processing was done locally on the machine where the user was interacting with the application. They also were in no way, shape, or form as "flashy" as the Visual Basic, PowerBuilder, Delphi, and C++ "fat clients" that followed them (which are still used today, although less so with the advent of webapps).

The webapps that have been built for many years now—indeed are still built today on a daily basis—have one big problem: they are, by and large, redrawing the entire screen each time some event occurs. They are intrinsically server-centric to a large extent. When the user does something (beyond some trivial things that can occur client side such as mouse-over effects and such), a server must get involved. It has to do some processing, and then redraw what the user sees to incorporate the applicable updated data elements. This is, as I'm sure you have guessed, highly inefficient.