CHAPTER 6

Don't Just Live in the Moment: Client-Side Persistence

For applications, two types of data storage are available: *persistent* and *nonpersistent*, or transient. Persistent storage is any storage mechanism that provides a place for data to be saved between program executions, and often for an indefinite period of time (until explicitly removed from storage). Transient storage is any storage mechanism where the data lives only as long as the program is actually executing (or for some short time thereafter). A database is generally considered a persistent storage mechanism, whereas RAM clearly is not. Writing to a hard drive is usually persistent as well, while session memory generally is not. The term *durable* is also often used to describe persistent storage media.

In web applications, storing state on the server—whether in a database or in a session—is pretty much expected of most applications. But what happens when you don't have a persistence mechanism on the server, or possibly don't want one for some reason? What's the alternative?

When discussing persistence in a pure JavaScript-based client-side application, there is a very short list of possible storage mechanisms.[1] The ubiquitous cookie—small bits of information stored on the client on a per-domain basis—is one. Another is a facility now available on most browsers, dubbed *local shared objects* (or Flash shared objects, or even Flash cookies, depending on the documentation you read). This is a storage mechanism provided by the Adobe Flash browser plug-in.

Local shared objects are gaining quite a bit of popularity, and we'll put that approach to use in this chapter's project, which is a simple contact manager. Perhaps most interesting though is the Dojo library we will use to implement this client-side persistence, which makes our lives so much easier. So, on with the show.

Contact Manager Requirements and Goals

In this chapter, we will build a Contact Manager application that runs entirely on the client side. This is handy because it means it can run on virtually any PC, without requiring a network connection (although, conversely, it means that the contact data can't easily be shared among

1. Java applets or ActiveX controls are other options, but they are generally thought of as a whole other class of possibilities, because they can, in a sense, be seen as extensions to the browser itself (certainly, ActiveX controls are meant to be that, but applets can also be viewed in that light). In either case, they require something more than HTML and JavaScript, and that in and of itself places them in a different category.

multiple machines). Clearly, persistence will come into play here, since it would be quite a useless application if we couldn't actually store contacts.

Let's list some of the key things this project will accomplish and some of the features we'll seek to implement.

- Each contact should include a good amount of data. Along with the basics of name, phone number, and email address, we'll also allow for a fair amount of extended information, such as birthday, spouse's name, children's names, and so on. However, we want to make these items as free-form as possible, so users can use the different data fields however they like.
- We'll implement the typical alphabet selector tabs to make it easier to find contacts.
- We'll store our contacts with local shared objects.
- We'll use Dojo to provide some widgets coolness to make the interface fun and attractive. We'll also use Dojo to deal with the underlying details of working with local shared objects.

Now that we have some goals in mind, let's take a look at the library we'll use for this project.

Dojo Features

To help build the Contact Manager application, we'll use the very popular Dojo toolkit. Chapter 2 briefly introduced Dojo, but let's look at it in just a little more detail.

On its front page, Dojo (`http://dojotoolkit.org`) explains what it is, and I see no need to try to paraphrase, so here's a direct quote:

> *Dojo is the Open Source JavaScript toolkit that helps you build serious applications in less time. It fills in the gaps where JavaScript and browsers don't go quite far enough, and gives you powerful, portable, lightweight, and tested tools for constructing dynamic interfaces. Dojo lets you prototype interactive widgets quickly, animate transitions, and build Ajax requests with the most powerful and easiest to use abstractions available. These capabilities are built on top of a lightweight packaging system, so you never have to figure out which order to request script files in again. Dojo's package system and optional build tools help you develop quickly and optimize transparently.*
>
> *Dojo also packs an easy to use widget system. From prototype to deployment, Dojo widgets are HTML and CSS all the way. Best of all, since Dojo is portable JavaScript to the core, your widgets can be portable between HTML, SVG, and whatever else comes down the pike. The web is changing, and Dojo can help you stay ahead.*
>
> *Dojo makes professional web development better, easier, and faster. In that order.*

Yes, that does about sum it up! Dojo has been gaining a following of late, and has even been integrated into some popular frameworks, such as WebWork from OpenSymphony (now Struts 2 from Apache, `http://www.opensymphony.com/webwork` or `http://struts.apache.org`).

Dojo is a rather large library, containing a myriad of packages and features. Dojo is not just about Ajax, like some other libraries out there. It provides some functions that extend JavaScript itself, as well as general utility code for JavaScript applications.