CHAPTER 8

Learning Ajax/JavaScript Widgets and Their APIs

In the previous two chapters, you studied web APIs, first that of Flickr and then of other applications. I showed you how to call APIs using REST, SOAP, and XML-RPC interfaces from PHP and Python. In this chapter, I'll begin an analysis of one extremely important context where web APIs are used: JavaScript inside the modern web browser—the stuff called Ajax.

The term Ajax was coined as shorthand for *Asynchronous JavaScript and XML*. In Chapter 10, I'll show you some of the underlying flow of data of Ajax when you learn how to fully exercise the Flickr API from JavaScript to create a mashup. In this chapter, I'll teach you how to use Ajax *widgets*, JavaScript-based programs created by others to express some functionality, as a way to a study of Ajax. Along the way, you'll learn how to use some debugging tools such as Firebug and the JavaScript Shell that will help you make sense of these widgets and, as I'll show you later, the whole range of Ajax programming.

In the context of contemporary Web 2.0 development, Ajax is a big deal, particularly for how it allows you to mash up data and services in new and easier ways. Ajax exploits the fact that modern web browsers are programmable and that they are inherently network applications. In addition to sending static HTML to web browsers, programmers can send JavaScript programs to run in the web browser. What can be done with this type of JavaScript-based client-side programming?

* You can achieve more dynamic interaction without having to reload the entire web page. This capability can be used, for instance, for drop-down menus and other widgets that we are used to having on the desktop.
* In particular, JavaScript can be used to get data via formal and informal web APIs from a server without having to reload the entire web page.
* Widgets can be created and deployed by other people. These widgets can be used to combine data and services and shown to other people. (Google Maps is the single most mashed-up API/service on the public Internet.)

JavaScript and DHTML are not new phenomena but have become extremely popular under the banner of Ajax. Jesse James Garrett says it well:[1]

But seeing Ajax as a purely technological phenomenon misses the point. If anything, Ajax is even more of a sea change for designers than it is for developers. Sure, there are a lot of ways in which developers need to change their thinking as they make the transition from building traditional web applications to building Ajax applications. But for those of us who *design user experiences, the change brought about by Ajax is even more profound.*

This chapter concentrates on helping you use Ajax to mash up data and services by doing the following:

- Pointing out the Ajax-based parts of Flickr and contrasting the old style of web development that involved the reloading of an entire page to new-style development in which more logic is pushed to the client, opening up more opportunities for integration
- Pointing out ways to see the difference between Ajax and non-Ajax apps by turning off JavaScript in the browser
- Introducing the Yahoo! UI Library as a specific example of various JavaScript widget libraries
- Introducing Google Maps, the single most used API as an example of a JavaScript widget
- Using one of the JavaScript widget libraries to demonstrate how to use a widget (for example, the TreeView widget)
- Showing how to write a basic Greasemonkey script as a way of mashing up services and data in the browser

What You Need to Know

Ajax, along with all its attendant use of JavaScript and the modern web browser, is a rich subject, as can be seen in the myriad of books that have been published recently on the subject. I'll put Ajax in the larger context of the programmable web browser. To become a master programmer of the web browser, you should understand the following:

- Both how an ideal W3C DOM standards-compliant browser works and how various browsers actually work in various areas: how JavaScript is implemented, object model behavior, CSS, and events
- JavaScript-based APIs and widgets such as Google Maps—what they are and how to use them
- Nonbrowser environments for JavaScript, such as Google Gadgets, Yahoo! Widgets, and Adobe Acrobat

1. Ajax Hacks by Bruce Perry (O'Reilly & Associates, 2006)

- Extension mechanisms in browsers (such as Firefox add-ons)
- JavaScript and browser debugging tools such as Firebug
- JavaScript libraries: how they relate and what can be intermixed—and which ones are tied to which web programming frameworks
- What people have done already on all these fronts using JavaScript and remixing the browser

* How to write JavaScript and JavaScript widgets that can be reused by other people, including cross-platform JavaScript
* What you can do in terms of mashups

Fortunately, you do not need to know all these things to merely get started.

What Difference Does Ajax Make?

To convince yourself that JavaScript is at work in web applications such as Flickr, Google Maps, and Gmail, you can turn off JavaScript in your browser and see what changes in the behavior of the application.

To turn off JavaScript in your browser, do the following, depending on which browser you're using:

* In Firefox, uncheck Tools ~TRA Options ~TRA Content ~TRA Enable JavaScript.
* In Internet Explorer, check Tools ~TRA Internet Options ~TRA Security ~TRA Custom Level ~TRA Scripting ~TRA Active Scripting ~TRA Disable.
* In Opera, uncheck Tools ~TRA Quick Preferences ~TRA Enable JavaScript.
* In Safari, uncheck Safari ~TRA Preferences ~TRA Security ~TRA Enable JavaScript.

Once you have JavaScript turned off, notice the following changes in Flickr and Google Maps:

* In Flickr pages for a specific photo (in other words, `http://flickr.com/photos/{user-d}/{photo-id}/`), you will see the message "To take full advantage of Flickr, you should use a JavaScript-enabled browser and install the latest version of the Macromedia Flash Player." All the buttons on top of the picture no longer function. Instead of clicking the title, description, and tags to start editing them, you have to click a link (Edit Title, Description, and Tags) before doing so.
* Notice that some apps will gracefully support non-JavaScript-enabled browsers—particularly Google Maps. With JavaScript turned off, you no longer see the pan and zoom new-style maps but an old-style map that provides links to move north, south, east, or west or to change the zoom level.

When using JavaScript, there are interesting and important issues regarding usability/accessibility. Many computers, including mobile devices, do not use JavaScript. How should apps gracefully deal with browsers that don't use JavaScript? Some apps are so dependent on JavaScript that a non-JavaScript version would look drastically different.

Now that you have seen the effects of turning off JavaScript in the browser, be sure to turn it back on if you want to learn how to use JavaScript-based widgets and APIs.

Learning Firebug, DOM Inspector, and JavaScript Shell

In learning Ajax and widgets/applications based on Ajax, I recommend using Firefox, the DOM Inspector, and the JavaScript Shell (bookmarklet) to manipulate live web pages and widgets. This combination allows for live interaction with the little apps; you can load a running map, analyze the details of how it is working while running, and issue commands that take immediate effect.

Note Even if you work primarily in a web browser other than Firefox, you can still gain a tremendous amount of insight about JavaScript programming using the Firefox-based tools.

Moreover, the right tools can help you make sense of complicated stuff such as the Browser Object Model and the Document Object Model (DOM) by letting you interact with the browser, test code, and so on. I will use these tools in the rest of the chapter to support this experimental/reverse-engineering approach.

Using the DOM Inspector

The DOM Inspector allows you to look at the HTML DOM as a tree and make changes in that DOM. The DOM Inspector comes with Firefox but is not installed by default in Windows. To install it in Windows, you need to choose explicitly to install the DOM Inspector in the installation process. Consult the following documentation to learn how to use the DOM Inspector:

```
http://kb.mozillazine.org/DOM_Inspector
```

Remember, you can invoke the DOM Inspector by selecting Tools ~□□□ DOM Inspector from the Firefox menu.

Using the Firebug Extension for Firefox

I highly recommend installing the Firefox add-on called Firebug. In many ways, Firebug is an enhancement of the DOM Inspector. In addition to being able to view and edit the contents of the DOM, you also get some very nice JavaScript debugging functionality.

To install this extension in Firefox, navigate to `http://getfirebug.com`, give permission to Firefox to install the extensions from that domain, and then install the extension. (As with installing all Firefox add-ons, you need to restart the browser to complete the installation.)

Figure 8-1 shows Firebug when I was using its Inspect HTML functionality to examine the title of a Flickr photo.

Insert 858Xf0801.tif

Figure 8-1. Firebug applied to a Flickr photo page. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)

Important features of Firebug include the following:

* The ability to view live source (that is, what the HTML of the DOM is at the moment, not what the original source was).
* Instant HTML editing (you can edit the HTML in Firebug and see the changes reflected on the page).
* You can see the request and response headers, which is invaluable during debugging.

Firebug can therefore be useful for the following tasks:

* Learning CSS by changing `<style>` elements and seeing the effects (for example, the cascading process is shown, and overridden properties are struck out).
* Tracking uses of the `XMLHttpRequest` object (the object, which we will see in the next chapters, is the one often responsible for the exchange of XML or JSON by the browser). You can use Firebug to see whether and what data is actually being loaded.
* Using inspect mode to mouse over a piece of a web page and see the corresponding HTML.

Using the JavaScript Shell

The JavaScript Shell is a bookmarklet to use in Firefox that you can find at `https://www.squarefree.com/bookmarklets/webdevel.html`. A *bookmarklet* is a short piece of JavaScript that you can treat like a browser bookmark but that does some function when you click it. With the JavaScript Shell, you can run snippets of JavaScript code that will execute in the context of the page in which you invoked the shell.

To install the JavaScript Shell, just drag it to your Links toolbar in Firefox. To invoke the JavaScript Shell, put the page you want in the foreground, and click the JavaScript Shell bookmarklet.

Note In addition to the tools already mentioned, there is Firebug Lite to use with Internet Explorer, Opera, and Safari.[2] You might consider using the Venkman JavaScript Debugger.[3]

Working with JavaScript Libraries

Instead of programming directly for a specific browser, it is often useful to work at a higher level of abstraction by working with a JavaScript library. There are many cross-browser differences and fine technical details that are best left to the JavaScript specialist. JavaScript libraries typically allow you to program the browser as a generic entity rather than having to account for the differences among browsers.

Ideally there would be one obvious choice for an excellent JavaScript library, and everyone would use it. The current situation is that there are many JavaScript libraries, and it is not at all obvious how they compare. For instance, Simon Willison, a well-respected web developer, wrote that the big four are the following:[4]

* Dojo
* Mochikit
* Prototype/script.aculo.us
* Yahoo! UI Library (YUI)

Others have pointed out Rico (which is built on top of Protoype) and OpenLaszlo.[5] In this chapter and those that follow, I will concentrate on using YUI.

2. http://www.getfirebug.com/lite.html
3. http://www.mozilla.org/projects/venkman/
4. http://simonwillison.net/2006/Jun/26/libraries/
5. http://www.openlaszlo.org/

YUI Widgets

You can find the Yahoo UI Library at `http://developer.yahoo.com/yui/` where you can read "Yahoo! UI Library—Getting Started."[6] The best way is to learn about the library is to look around and try the various examples. Also use the JavaScript Shell and Firebug extension to learn how things work.

Try the pieces on the Yahoo! site (for example, the TreeView controller at `http://developer.yahoo.com/yui/examples/treeview/index.html`), and enter some commands on the JavaScript Shell. To help you out, we will walk through the use of two YUI widgets: the calendar and the TreeView widget.

Using the YUI Calendar

The YUI Calendar component (`http://developer.yahoo.com/yui/calendar/`) presents a browser-based calendar interface from which users can select one or more dates. To learn how to use it, you can try the calendar examples:

* `http://developer.yahoo.com/yui/examples/calendar/index.html` (refer to the API documentation)
* `http://developer.yahoo.com/yui/docs/YAHOO.widget.Calendar.html` (to get a list of methods for the widget)

You can use the Firebug extension or JavaScript Shell to get a feel for how to program the component:

1. Go to `http://developer.yahoo.com/yui/examples/calendar/quickstart.html`.
2. Click a date.
3. In the console of Firebug or the JavaScript Shell, type the following to get back the date you selected (see Figure 8-2):

```
YAHOO.example.calendar.cal1.getSelectedDates()
```

4. Try other methods to see how the calendar works:

```
YAHOO.example.calendar.cal1.hide() // Hides the control
YAHOO.example.calendar.cal1.show() // Shows the control
// Sets month to February (change not visible until redrawn)
YAHOO.example.calendar.cal1.setMonth(1)
// Redraws the control using the YUI TreeView
YAHOO.example.calendar.cal1.render()
```

6. http://developer.yahoo.com/yui/#start

Insert 858Xf0802.tif

Figure 8-2. Interacting with the Yahoo! Calendar using the JavaScript Shell. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)

The TreeView component is a UI control that lets users interact with a tree structure (by, for instance, expanding or collapsing branches of the tree):

```
http://developer.yahoo.com/yui/treeview/
```

In addition to reading the API documentation for the TreeView component,[7] you can use Firebug and the JavaScript Shell to experiment with the component. To do so, follow these steps:

1. Go to `http://developer.yahoo.com/yui/examples/treeview/default_tree.html`.
2. In the console of Firebug or the JavaScript Shell, type the following to expand and collapse the tree, respectively:

```
tree.expandAll()
tree.collapseAll()
```

See how you can interactively learn how to use the YUI JavaScript widgets through using Firebug and the JavaScript Shell.

Installing YUI on Your Host

To use YUI in your own applications, you should set up YUI on your own web host. I will use a concrete example, `http://examples.mashupguide.net`, which is mapped to the Unix directory `~/examples.mashupguide.net`. Substitute your own values. My goal is to set up YUI so that it is accessible at `http://examples.mashupguide.net/lib/yui/`.

7. http://developer.yahoo.com/yui/docs/YAHOO.widget.TreeView.html

1. Download the library to your machine or your web hosting environment. Go to `http://developer.yahoo.com/yui/download/`.
2. Unzip and copy the files to the right location. In my case, I unzipped and copied the unzipped directory (which is named `yui`) to `/home/rdhyee/examples.mashupguide.net/lib/yui`, which maps to the `yui` directory at `http://examples.mashupguide.net/lib/yui/`. The important part of the library for runtime purposes is the `yui/build` directory.

With the files in your own directory, you can, for instance, look at the calendar example on my own server:

```
http://examples.mashupguide.net/lib/yui/examples/calendar/quickstart.html
```

Learning Google Maps

Just as there are different UI elements packaged up in one of the major JavaScript libraries, vendors have already started to create reusable JavaScript components. The most famous of these Ajax widgets is Google Maps. In this section, we'll look at how to embed a Google map using the Google Maps API. The online documentation on how to get started with the maps at the Google web site is good.[8] The approach given there, one that I can certainly recommend, is to give you source code for increasingly more complex examples, which you can copy and paste to your own site.

Here we will set up a simple map and then use the JavaScript Shell to work with a live map so that you can invoke a command and see an immediate response. The intended effect is that you see the widgets as dynamic programs that respond to commands, whether that command comes in a program or from you entering that command.

Use the Google Maps API to make a simple map:

1. Make sure you have a public web directory to host your map and know the URL of that directory. Any Google map that uses the free, public API needs to be publicly visible.
2. Go to the sign-up page for a key to access Google Maps.[9] You will need a key for any given domain in which you host Google Maps. (It is through these keys that Google regulates the use of the Google Maps API.)
3. Read the terms of service,[10] and if you agree to them, enter the URL directory on the host where you want to place your test file. For example, in my case, the URL is `http://examples.mashupguide.net/ch08`. Write down the resultant key.
4. Copy and paste the HTML code into your own page in your web-hosting directory. You should get something like the following (except that the code will have your API key):[11]

8. http://www.google.com/apis/maps/documentation/#Introduction

9. http://www.google.com/apis/maps/signup.html

10. http://www.google.com/apis/maps/terms.html

11. http://maps.google.com/maps/api_signup?url=http%3A%2F%2Fexamples.mashupguide.net%2Fch08

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script
```

```
      src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=[API_key]"
      type="text/JavaScript"></script>
    <script type="text/JavaScript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

5. Now make one modification to the example by removing the `var` keyword in front of `map` to make it a global variable that is thus accessible to the JavaScript Shell. That is, change the following:

```
var map = new GMap2(document.getElementById("map"));
to map = new GMap2(document.getElementById("map"));
```

   to expose the `map` object to the JavaScript Shell utility.[12]

5. Invoke the JavaScript Shell for your map by hitting the JavaScript Shell bookmarklet in the context of your map. Type the following code fragments, and see what happens. (Note that another approach is to modify your code directly with these code fragments and reload your page.) One can correlate the actions to the documentation for version 2 of the Google Maps API.[13]

12. http://examples.mashupguide.net/ch08/google.map.html

13. http://www.google.com/apis/maps/documentation/reference.html#GMap2

To return the current zoom level of the map (which goes from 0 to 17, with 17 for the most detailed), use this:

```
map.getZoom()
```

```
13
```

To obtain the latitude and longitude of the center of the map, enter the following:

```
map.getCenter()
```

```
(37.4419, -122.1419)
```

To center the map around the Campanile for UC Berkeley, use this:

```
map.setCenter(new GLatLng(37.872035,-122.257844), 13);
```

You can pan to that location instead:

```
map.panTo(new GLatLng(37.872035,-122.257844));
```

To add a small map control (to control the zoom level), use the following:

```
map.addControl(new GSmallMapControl())
map.addControl(new GMapTypeControl())
```

To add GMap keyboard navigation so that you can pan and zoom with the keyboard, use this:

```
window.kh = new GKeyboardHandler(map)
```

```
[object Object]
```

To fully zoom out the map, use this:

```
map.setZoom(0)
```

To zoom in all the way, use the following:

```
map.setZoom(17)
```

To set the variable `maptypes` to an array holding three objects, use this:

```
maptypes = map.getMapTypes()
```

```
[object Object],[object Object],[object Object]
```

To get the name of the first entry in `maptypes` (`1` corresponds to satellite, while `2` corresponds to the hybrid map type):

```
map.getMapTypes()[0].getName()
```

```
Map
```

To get the current map type, you can get the object and the name of that type object:

```
map.getCurrentMapType()
```

```
[object Object]
```

```
map.getCurrentMapType().getName()
```

```
Map
```

To set the map type to satellite, use the following:

```
map.setMapType(maptypes[1]);
```

You can zoom one level in and out if you are not already at the max or min zoom level:

```
map.zoomIn()
map.zoomOut()
```

To make an overlay, try the following:

```
point = new GLatLng (37.87309185260284, -122.25508689880371)
```

```
(37.87309185260284, -122.25508689880371)
```

```
marker = new GMarker(point)
```

```
[object Object]
```

```
map.addOverlay(marker);
```

To make something happen when you click the marker, type the following:

```
GEvent.addListener(marker, 'click', function() {
  marker.openInfoWindowHtml('hello'); })
```

```
[object Object]
```

There are many more features to explore, such as polylines, overlays, and draggable points. To learn more, I certainly recommend the "Google Maps API: Introduction" documentation.[14] I will also return to the topic of Google Maps in Chapters 10 and 13.

14. http://www.google.com/apis/maps/documentation/#Introduction

Accessing Flickr via JavaScript

In Chapter 10, we will be building a mashup that integrates Flickr data and Google Maps within the browser context. That is, we will need to call the Flickr API from JavaScript within the browser (in true Ajax style). Flickr provides JSON output to its API.[15]

In Chapter 4, I already presented some code that reads a Flickr feed in JSON format and renders it in HTML. In this section, I'll show you one basic way to use this JSON data from the Flickr API via JavaScript. Let's jump into how it works:

1. Go to the `flickr.photos.search` page in the Flickr Explorer (`http://www.flickr.com/services/api/explore/?method=flickr.photos.search`), set the `tag` parameter to `flower`, and set the `per_page` parameter to `3` (to make for a more manageable number of photos for now). Do not sign the call. Hit the Call Method button, and grab the REST URL (below the results box). Substitute the `api_key` parameter with your own Flickr API key. You will get something like this:

```
http://api.flickr.com/services/rest/?method=flickr.photos.search&~CCC
api_key=<API_key>&tags=flower&per_page=3
```

from which you get the Flickr XML output with which you are familiar from previous chapters.

2. Let's now study the JSON output by tacking on the parameter `format=json` to the URL.[16] Now you get the following (in a prettified rendition):

```
jsonFlickrApi( {
   "photos" : {
      "page" : 1, "pages" : 283266, "perpage" : 3, "total" : "849797",
      "photo" : [ {"id" : "397677823", "owner" : "28374750@N00",
      "secret" : "cab3f3db01", "server" : "124", "farm" : 1, "
      title" : "DSC_0026", "ispublic" : 1, "isfriend" : 0, "isfamily" : 0}
      , {
         "id" : "397677820", "owner" : "28374750@N00",
         "secret" : "f70cf0bb19", "server" : "174", "farm" : 1,
         "title" : "red flowers", "ispublic" : 1, "isfriend" :
0, "isfamily" : 0}
      , {
         "id" : "397677553", "owner" : "37015070@N00",
         "secret" : "7329c71748", "server" : "158", "farm" : 1,
         "title" : "Rose In Vase", "ispublic" : 1, "isfriend" : 0,
         "isfamily" : 0}
      ]}
   , "stat" : "ok"}
)
```

Note what is being returned: a piece of JavaScript containing the function named `jsonFlickrApi()` with a single parameter, which is a JavaScript object that represents the photos.

15. http://www.flickr.com/services/api/response.json.html

16. http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key={api-key}&tags=flower&per_page=3&format=json

3. Let's now write a bit of HTML and JavaScript to test how to write JavaScript to access the various pieces of the Flickr response:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Flickr JSON</title>
  </head>
<script>
function jsonFlickrApi(rsp) {
  window.rsp = rsp;
}
</script>
<script
src="http://api.flickr.com/services/rest/?method=flickr.photos.search~CCC
&api_key=<API_key>&tags=flower&per_page=3&format=json"></script>
</html>
```

You can load this page in your browser and invoke the JavaScript Shell to learn a few key lines to access parts of the response:

```
props(rsp.photos)
Fields: page, pages, perpage, photo, total
rsp.photos.perpage
3
rsp.photos.photo[0].id
397694840
```

4. Let's now have `jsonFlickrApi()` produce a display of the photos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Flickr JSON</title>
  </head>
<body>
  <script>
  function jsonFlickrApi(rsp) {
    window.rsp = rsp;
    var s = "";
    // http://farm{id}.static.flickr.com/{server-id}/{id}_{secret}_[mstb].jpg
    // http://www.flickr.com/photos/{user-id}/{photo-id}
    s = "total number is: " + rsp.photos.photo.length + "<br/>";

    for (var i=0; i < rsp.photos.photo.length; i++) {
      photo = rsp.photos.photo[i];
      t_url = "http://farm" + photo.farm + ".static.flickr.com/" +
        photo.server + "/" + photo.id + "_" + photo.secret + "_" + "t.jpg";
      p_url = "http://www.flickr.com/photos/" + photo.owner + "/" + photo.id;
      s +=  '<a href="' + p_url + '">' + '<img alt="'+ photo.title +
        '"src="' + t_url + '"/>' + '</a>';
    }
    document.writeln(s);
  }
  </script>
  <script src="http://api.flickr.com/services/rest/?method=flickr.photos. ~CCC
search&api_key=<API_key>&tags=flower&per_page=50&format=json"></script>
</body>

</html>
```

5. The previous example is simple but limited by the fact that loading JSON data immediately calls the function `jsonFlickrApi()`. You can customize the name of the callback function with the `jsoncallback` parameter (for example, `jsoncallback=MyHandler`). You can also use the `nojsoncallback=1` parameter[17] to return raw JSON:

```
{
   "photos" : {
      "page" : 1, "pages" : 283353, "perpage" : 3, "total" : "850057",
      "photo" : [ {"id" : "397750427", "owner" : "98559475@N00",
      "secret" : "f59b1ae9e1", "server" : "180", "farm" : 1,
      "title" : "sakura", "ispublic" : 1, "isfriend" : 0, "isfamily" : 0}
      , {
         "id" : "397750433", "owner" : "81222973@N00",
         "secret" : "0023e79dff", "server" : "133", "farm" : 1,
         "title" : "just before spring", "ispublic" : 1,
         "isfriend" : 0, "isfamily" : 0}
      ]}
   , "stat" : "ok"}
```

Given this information, you might be wondering why should you use the `jsonFlickrApi()` callback in the first place instead of using a pattern based on the examples I showed you in Chapter 6 of how to use PHP to interact with the Flickr API. That is, why not use JavaScript to do the following:

1. Do an HTTP GET on this:

```
http://api.flickr.com/services/rest/?method=flickr.photos.search~CCC
&api_key={api-key}&tags=flower&per_page=3&format=json&nojsoncallback=1
```

2. Instantiate the response as a JavaScript object that you can then parse at your leisure.

I will, in fact, walk you through such an approach in Chapter 10. I defer the discussion until then because we will need to deal with security issues particular to web browsers in order to make our examples work.

17. http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key={api-key}&tags=flower&per_page=3&format=json&nojsoncallback=1

Using Greasemonkey to Access *New York Times* Permalinks

Greasemonkey is an add-on for Firefox that allows you to change the behavior of web pages in your browser. That includes creating mashups. You already saw an example of a Greasemonkey script in Chapter 1—the Google Maps in Flickr Greasemonkey script. And here are some good references you can use to get the best out of Greasemonkey:

* `http://diveintogreasemonkey.org/`
* *Greasemonkey Hacks* by Mark Pilgrim (O’Reilly Media, 2005)
* `http://en.wikipedia.org/wiki/Greasemonkey`
* `http://www.greasespot.net/` (the official blog of the Greasemonkey project)

Links that show up on the *New York Times* online site typically expire after a week. That is, instead of going to the article, you are given an excerpt and a chance to purchase a copy of the article. However, in 2003, Dave Winer struck a deal with the *New York Times* to provide a mechanism to get weblog-safe permalinks to articles.[18] Aaron Swartz wrote a *New York Times* link generator that compiles those permalinks and makes them available for lookup via a web form or a JavaScript bookmarklet.[19] That is, you give it a URL to a *New York Times* article, and it will return to you a more permanent link.

Let's look at an example. Consider the following URL:

```
http://www.nytimes.com/2007/04/04/education/04colleges.html
```

This corresponds to the following:

```
http://www.nytimes.com/2007/04/04/education/04colleges.html?ex=1333339200&~CCC
en=3b7aac16a1ce4512&ei=5090&partner=rssuserland&emc=rss
```

You can see this for yourself by going to the link generator:

```
http://nytimes.blogspace.com/genlink?q=http://www.nytimes.com/2007/04/04/educati
on/~CCC
04colleges.html
```

When there is no permalink for an article, you will see a different type of output from the *New York Times* link generator. For example, consider the following:

```
http://www.nytimes.com/aponline/us/AP-Imus-Protests.html
```

18. http://www.scripting.com/davenet/2003/06/06/newYorkTimesArchiveAndWebl.html

19. http://nytimes.blogspace.com/genlink

This doesn't have a permalink, as you can see from this:

```
http://nytimes.blogspace.com/genlink?q=http://www.nytimes.com/aponline/us/~CCC
AP-Imus-Protests.html
```

Where's a good place to stick a UI element for the permanent link on the *New York Times* page? There are lots of choices, but a good one is a toolbar with such elements as e-mail/print/single-page/save/share.

The basic logic of the Greasemonkey script we want to write consists of the following:

1. If you are on a *New York Times* article, send the link to the *New York Times* link generator.
2. If there is a permalink (which you will know is true if the `href` attribute of the first `<a>` tag starts with `http:` and not `genlink`), insert a new `<li>` element at the end of the `<ul id="toolsList">`.

Now let's walk through the steps to get this functionality working in your own Firefox browser installation:

1. Install the Greasemonkey extension if you don't already have it installed.[20]
2. Create a new script in Greasemonkey in one of two ways:

   a. You can go to `http://examples.mashupguide.net/ch08/newyorktimespermalinker.user.js` and click Install.

   b. Select Tools ***~TRA*** Greasemonkey ***~TRA*** New User Script, fill in Name/Namespace/Description/Includes, and then enter the following code. (Note the use of `GM_xmlhttpRequest` to find out what a more permanent link is. You will see in Chapter 10 the logic behind `xmlhttpRequest`.[21])

```
// ==UserScript==
// @name           New York Times Permlinker
// @namespace      http://mashupguide.net
// @description    Adds a link to a "permalink" or  "weblog-safe" URL
// for the NY Times article, if such a link exists
// @include        http://*.nytimes.com/*
// ==/UserScript==

function rd(){

  // the following code is based on the bookmarklet written by Aaron Swartz
  // at http://nytimes.blogspace.com/genlink

  var x,t,i,j;
  // change %3A -> : and %2F -> '/'
  t=location.href.replace(/[%]3A/ig,':').replace(/[%]2f/ig,'/');

  // get last occurrence of "http://"
  i=t.lastIndexOf('http://');

  // lop off stuff after '&'
  if(i>0){
    t=t.substring(i);
    j=t.indexOf('&');
    if(j>0)t=t.substring(0,j)
  }

  var url = 'http://nytimes.blogspace.com/genlink?q='+t;

  // send the NY Times link to the nytimes.blogspace.com service.
  // If there is a permalink, then the href attribute of the first tag
  //  will start with 'http:' and not 'genlink'.
  // if there is a permalink, then insert a new li element at the end of the
  // <ul id="toolsList">.

  GM_xmlhttpRequest({
  method:"GET",
  url:url,
  headers:{
    "User-Agent":"monkeyagent",
    "Accept":"text/html",
  },
  onload:function(details) {
    var s = details.responseText;
    var p = /a href="(.*)"/;
    var plink = s.match(p)[1];
    if ( plink.match(/^http:/) &&
         (tl = document.getElementById('toolsList')) )  {
      plink = plink + "&pagewanted=all";
      plinkItem = document.createElement('li');
      plinkItem.innerHTML = '<a href="' + plink + '">PermaLink</a>';
      tl.appendChild(plinkItem);
    }
  }
});

}

rd();
```

3. What you will see now with this Greasemonkey script if you go to `http://www.nytimes.com/2007/04/04/education/04colleges.html` is the new entry Permalink underneath the Share button. The link may take a few seconds to appear while the permalink is retrieved.

Note that this Greasemonkey script is sensitive to changes in the way *New York Times* articles are laid out: older articles have a different page structure and therefore need some other logic to put the permalink in the right place.

20. https://addons.mozilla.org/en-US/firefox/addon/748

21. http://wiki.greasespot.net/GM_xmlhttpRequest and http://diveintogreasemonkey.org/api/
gm_xmlhttprequest.html

Learning More About JavaScript and Ajax

It's good to have some handy references on JavaScript programming while working on Ajax-related projects:

* *JavaScript, the Definitive Guide, Fifth Edition* by David Flanagan (O'Reilly Media, 2006)
* *Pro JavaScript Techniques* by John Resig (Apress, 2006)
* *Learning JavaScript* by Shelley Powers (O'Reilly Media, 2007)

I'm particularly fond of Peter-Paul Koch's web site, which has tons of useful resources:

* The general resource page.[22] See also his introduction to JavaScript.[23]
* He has all the example scripts from his book (`http://www.quirksmode.org/book/`) that you can study at `http://www.quirksmode.org/book/examplescripts.html`.

Summary

In this chapter, you learned about how Ajax has changed the style of programming in contemporary web applications. You did so by turning off JavaScript in your browser and seeing what happens to Flickr and Google Maps. You studied the functionality of a JavaScript library (Yahoo! UI Library) and Google Maps via the Firebug Firefox extension and the JavaScript Shell. You also learned the basics of invoking the Flickr API in Ajax. Finally, you learned how to create a basic Greasemonkey script that inserts a permanent link into a *New York Times* online article. Although there is much more to learn concerning JavaScript and Ajax, this chapter provides the necessary background to the following chapters.

22. http://www.quirksmode.org/resources.html

23 http://www.quirksmode.org/js/intro.html