Easy Ajax with Prototype

There's little more impressive on the web today than a appropriate touch of Ajax. Used well, Ajax brings a web interface much closer to the experience of a desktop app, and can turn a bear of an task into a pleasurable activity.

But it's really hard, right? It involves all the nasty JavaScript that no one ever does often enough to get really good at, and the browser support is patchy, and urgh it's just so much damn effort. Well, the good news is that – ta-da – it doesn't have to be a headache. But *man* does it still look impressive. Here's how to amaze your friends.

Introducing prototype.js

Prototype is a JavaScript framework by Sam Stephenson designed to help make developing dynamic web apps a whole lot easier. In basic terms, it's a JavaScript file which you link into your page that then enables you to do cool stuff.

There's loads of capability built in, a portion of which covers our beloved Ajax. The whole thing is freely distributable under an MIT-style license, so it's good to go. What a nice man that Mr Stephenson is – friends, let us raise a hearty cup of mulled wine to his good name. Cheers! **sluurrrrp**.

First step is to download the latest Prototype and put it somewhere safe. I suggest underneath the Christmas tree.

Cutting to the chase

Before I go on and set up an example of how to use this, let's just get to the crux. Here's how Prototype enables you to make a simple Ajax call and dump the results back to the page:

```
var url = 'myscript.php';
var pars = 'foo=bar';
var target = 'output-div';
var myAjax = new Ajax.Updater(target, url, {method: 'get', parameters: pars});
```

This snippet of JavaScript does a GET to `myscript.php`, with the parameter `foo=bar`, and when a result is returned, it places it inside the element with the ID `output-div` on your page.

Knocking up a basic example

So to get this show on the road, there are three files we need to set up in our site alongside `prototype.js`. Obviously we need a basic HTML page with `prototype.js` linked in. This is the page the user interacts with. Secondly, we need our own JavaScript file for the glue between the interface and the stuff Prototype is doing. Lastly, we need the page (a PHP script in my case) that the Ajax is going to make its call too.

So, to that basic HTML page for the user to interact with. Here's one I found whilst out carol singing:

```
<!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" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Easy Ajax</title>
    <script type="text/javascript" src="prototype.js"></script>
    <script type="text/javascript" src="ajax.js"></script>
</head>
<body>
    <form method="get" action="greeting.php" id="greeting-form">
        <div>
            <label for="greeting-name">Enter your name:</label>
            <input id="greeting-name" type="text" />
            <input id="greeting-submit" type="submit" value="Greet me!" />
        </div>
        <div id="greeting"></div>
    </form>
</body>
</html>
```

As you can see, I've linked in `prototype.js`, and also a file called `ajax.js`, which is where we'll be putting our glue. (Careful where you leave your glue, kids.)

Our basic example is just going to take a name and then echo it back in the form of a seasonal greeting. There's a form with an input field for a name, and crucially a DIV (`greeting`) for the result of our call. You'll also notice that the form has a submit button – this is so that it can function as a regular form when no JavaScript is available. It's important not to get carried away and forget the basics of accessibility.

Meanwhile, back at the server

So we need a script at the server which is going to take input from the Ajax call and return some output. This is normally where you'd hook into a database and do whatever transaction you need to before returning a result. To keep this as simple as possible, all this example here will do is take the name the user has given and add it to a greeting message. Not exactly Web 2-point-HoHoHo, but there you have it.

Here's a quick PHP script – `greeting.php` – that Santa brought me early.

```
<?php
     $the_name = htmlspecialchars($_GET['greeting-name']);
     echo "<p>Season's Greetings, $the_name!</p>";
?>
```

You'll perhaps want to do something a little more complex within your own projects. Just sayin'.

Gluing it all together

Inside our `ajax.js` file, we need to hook this all together. We're going to take advantage of some of the handy listener routines and such that Prototype also makes available. The first task is to attach a listener to set the scene once the window has loaded. He's how we attach an `onload` event to the `window` object and get it to call a function named `init()`:

```
Event.observe(window, 'load', init, false);
```

Now we create our `init()` function to do our evil bidding. Its first job of the day is to hide the submit button for those with JavaScript enabled. After that, it attaches a listener to watch for the user typing in the name field.

```
function init(){
     $('greeting-submit').style.display = 'none';
     Event.observe('greeting-name', 'keyup', greet, false);
}
```

As you can see, this is going to make a call to a function called `greet()` `onkeyup` in the `greeting-name` field. That function looks like this:

```
function greet(){
     var url = 'greeting.php';
     var pars = 'greeting-name='+escape($F('greeting-name'));
     var target = 'greeting';
     var myAjax = new Ajax.Updater(target, url, {method: 'get', parameters: 
pars});
}
```

The key points to note here are that any user input needs to be escaped before putting into the parameters so that it's URL-ready. The target is the ID of the element on the page (a DIV in our case) which will be the recipient of the output from the Ajax call.

That's it

No, seriously. That's everything. Try the example. Amaze your friends with your 1337 Ajax sk1llz.