1 AJAX

In short, AJAX is essentially a way for JavaScript to call the server—usually to provide some data, or to get some new data.

Now, obviously you could've done that before—with a regular web-form; the neat part about AJAX is that it does it *asynchronously*. What that means it hat it doesn't block and wait for a response. It also means, you don't have to refresh the page.

The usual web-form method is *synchronous*, since the whole page goes blank, the web-browser queries the server, waits for server's response (in the mean time, the *user* is waiting), and then displays a completely new page.

AJAX method spawns a new thread, that queries the server, waits for a response, and then notifies the web-browser when the response arrives. The user never has to wait for the page to refresh (since waiting for the response happens in a background thread).

1.1 Concept

Conceptually, AJAX applications aren't that different from 'regular'[1] websites, except now there's the added complication of code running on the client (i.e.: JavaScript).

The way it works is: In JavaScript (client side), you create a request object. Then via JavaScript (usually initiated by some user event) you call the server (usually with data from a form). You setup a 'callback' method which will be called when the request returns. When the request comes back, your callback method is called, at which point it can modify the HTML to indicate what the request was. That's about it.

1.2 Example

Just as in the 'usual' case, you still need stuff on the server side to respond to your clients. Except this time, the response can be relatively simple text or XML. So lets create a simple CGI script to act as a back-end:

```
#!/usr/bin/perl

print "Content-Type: text/text\n\n";

read(STDIN,$_,$ENV{CONTENT_LENGTH}) if $ENV{CONTENT_LENGTH} > 0;
$_ .= '&'.$ENV{'QUERY_STRING'} if $ENV{QUERY_STRING};
$_ =~ s/\%(..)/chr(hex($1))/sgie;

print "RESPONSE($_): ",scalar localtime(),"\n";
```

The above script just returns the current time (and whatever else you typed into the POST/GET requests).

[1] If there is such a thing as 'regular' on the web.

Now that we know what the server does, gets turn attention to the client, which in this case, is a lot more interesting:

```
<html>
<head>
    <title>AJAX Sample</title>
</head>

<script language="JavaScript">
<!--
// variable to track request.
var req;

// get the async call object.
function getXMLHttpRequestObj (){
    var r = false;
    // branch for native XMLHttpRequest object
    if(window.XMLHttpRequest) {
        try {
            r = new XMLHttpRequest();
        } catch(e) {
            r = false;
        }
    // branch for IE/Windows ActiveX version
    } else if(window.ActiveXObject) {
        try {
            r = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
            try {
                r = new ActiveXObject("Microsoft.XMLHTTP");
            } catch(e) {
                r = false;
            }
        }
    }
    return r;
}

// function to send whatever user says to the server.
function sendStuff() {
    // get user's query.
    var idField = document.getElementById("query");
```

```
    // grab whatever the user is trying to say.
    var queryStr = idField.value;

    // construct url.
    var url = "tim.cgi";

    // get new async request object.
    req = getXMLHttpRequestObj();

    if(!req)    // if can't get it, just don't do anything.
        return false;

    // do a post request on the server; async.
    req.open("POST", url, true);

    // how it should get back to us.
    req.onreadystatechange = callback;

    // send data as plain form submission.
    req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    // write the post data
    req.send("query=" + escape(queryStr));

    return false;
}

// callback method---when the server gets back to us.
function callback() {
    // check whether everything is fine.
    if (req.readyState == 4) {
        if (req.status == 200){
            // display profphreak's response.
            var res = document.getElementById("responseText");
            res.innerHTML = req.responseText;
        }
    }
}
// -->
</script>
<body>
    <h1>AJAX Sample</h1>
```

```
    <form onsubmit="return sendStuff(this)">
    <p>Query:
    <input type="text" id="query" value="" size="70" style="width:80%">
    <input type="submit" value="Stuff"></p>
    </form>

<script language="JavaScript">
<!--
// when page first loads, text field grabs focus.
document.getElementById("query").focus();
// -->
</script>

<p id="responseText"></p>

</body>
</html>
```

The code is essentially just a form that with a text field, that calls `sendStuff` javascript method whenever the "Stuff" button is clicked. The `sendStuff` method is the one that sets up the call.

Upon return of the request, the `callback` method is called, which just alters the `responseText` tag.

The code revolves around this `XMLHttpRequest` object. Depending on which browser you're using, you might need to use slightly different methods of getting it. A relatively safe way of getting it in `getXMLHttpRequestObj` method.

...and that's about it.