Using the XML HTTP Request object

This article was originally written in April 2002, I've decided to fix and update it as the objects finally seem to be getting some popularity. The 2002 version is still available, as are the September 2004 and August 2005 versions. This version January 2006.

Internet Explorer on Windows, Safari on Mac OS-X, Mozilla on all platforms, Konqueror in KDE, IceBrowser on Java, and Opera on all platforms including Symbian provide a method for client side javascript to make HTTP requests. From the humble begins as an oddly named object with few admirers, it's blossomed to be the core technology in something called AJAX [1].

The Object makes many things easier and neater than they other would be, and introduces some things that were otherwise impossible such as HEAD requests to see when a resource was last modified, or to see if it even exists. It makes your scripting options more flexible allowing for POST requests without having the page change, and opens up the possibility of using PUT, DELETE etc. These methods are increasingly used to provide richer Web Applications like G-Mail that use lower bandwidth and offer snappier user interaction.

Why *XML* HTTP Request object?

Whilst the object is called the XML HTTP Request object it is not limited to being used with XML, it can request or send any type of document, although dealing with binary streams can be problematical in javascript.

Creating the object

In Internet Explorer, you create the object using `new ActiveXObject("Msxml2.XMLHTTP")` or `new ActiveXObject("Microsoft.XMLHTTP")` depending on the version of MSXML installed. In Mozilla and Safari (and likely in future UA's that support it) you use `new XMLHttpRequest()` IceBrowser uses yet another method the `window.createRequest()` method.

This means that you need to show different script to different browsers, as what works in one, will error in another. The script below does this, and if it's not supported, the variable is set to false to allow for appropriate error messages and recovery with degrading to more normal HTTP transaction methods when the object isn't available. This degradation is important, even in IE the objects can often be blocked by slightly raised security settings (popular due to the commonly exploited holes of course). Where possible degrade, some approaches are talked about below, if you really can't, I'd recommend providing an alternative page aswell. GMail for example has said they'll be providing a less demanding version in the future, hopefully with no javascript at all, full degradation.

```
var xmlhttp=false;
/*@cc_on @*/
/*@if (@_jscript_version >= 5)
// JScript gives us Conditional compilation, we can cope with old IE versions.
// and security blocked creation of the objects.
```

```
 try {
  xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
 } catch (e) {
  try {
   xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (E) {
   xmlhttp = false;
  }
 }
@end @*/
if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
        try {
                xmlhttp = new XMLHttpRequest();
        } catch (e) {
                xmlhttp=false;
        }
}
if (!xmlhttp && window.createRequest) {
        try {
                xmlhttp = window.createRequest();
        } catch (e) {
                xmlhttp=false;
        }
}
```

How do I make a request?

Making a HTTP request is very simple. You tell the XML HTTP request object what sort of HTTP request you want to make and which url you want to request. Provide a function to be called when as the request is being made, and finally what, (if any) information you want sent along in the body of the request.

The following script makes a GET request for the relative url "text.txt" (relative to the calling page) It provides the function, which checks the `readyState` property each time it's called and when it has the value 4 - meaning the load is complete, it displays the `responseText` to the user with an alert.

```
 xmlhttp.open("GET", "test.txt",true);
 xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4) {
   alert(xmlhttp.responseText)
  }
 }
 xmlhttp.send(null)
```

Try the example.

Making a HEAD request

With a HEAD request, a server will only return the headers of a resource, rather than the resource itself, this means you can find out the Content-Type or Last-Modified of a document, without downloading it itself.

A typical HEAD request might return something like this:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Cache-Control: max-age=172800
Expires: Sat, 06 Apr 2002 11:34:01 GMT
Date: Thu, 04 Apr 2002 11:34:01 GMT
Content-Type: text/html
Accept-Ranges: bytes
```

```
Last-Modified: Thu, 14 Mar 2002 12:06:30 GMT
ETag: "0a7ccac50cbc11:1aad"
Content-Length: 52282
```

To make a HEAD request, you simply replace the first parameter with HEAD, and then extract the headers, either using `getAllResponseHeaders` or `getResponseHeader("Name")` to get an individual one.

```
xmlhttp.open("HEAD", "/faq/index.html",true);
xmlhttp.onreadystatechange=function() {
 if (xmlhttp.readyState==4) {
  alert(xmlhttp.getAllResponseHeaders())
 }
}
xmlhttp.send(null)
```

Try the example.

Using HEAD requests, to find the Last-Modified of another file.

One use of HEAD requests, is to find out when a url was modified, extending the previous example, you get something like this:

```
xmlhttp.open("HEAD", "/faq/index.html",true);
xmlhttp.onreadystatechange=function() {
 if (xmlhttp.readyState==4) {
  alert("File was last modified on - "+
   xmlhttp.getResponseHeader("Last-Modified"))
 }
}
xmlhttp.send(null)
```

Try the example.

To format the date differently, or use something other than alert, the javascript FAQ will tell you more.

Does a url exist?

Another simple use is finding if a url exists, in HTTP there are various status codes returned by both HEAD and GET requests, 200 means success, 404 means failure, and the others mean other things. See HTTP status codes for a full explanation. using the `status` property of the xmlhttp object provides you this status

```
xmlhttp.open("HEAD", "/faq/index.html",true);
xmlhttp.onreadystatechange=function() {
 if (xmlhttp.readyState==4) {
  if (xmlhttp.status==200) alert("URL Exists!")
   else if (xmlhttp.status==404) alert("URL doesn't exist!")
    else alert("Status is "+xmlhttp.status)
 }
}
xmlhttp.send(null)
```

Try the example: with a url that exists, with a url that does not exist

Calling a server-side Script without refreshing the page

Forms are the way to "call" serverside scripts in HTML, they force the page reload, and this is often not very user friendly. Using the HTTP Request, you can call the script without refreshing the page, and still have the form "fallback" to working when the XML HTTP Request Object is not available.

```
<%
 a=+(Request.QueryString('a')+'')
 b=+(Request.QueryString('b')+'')
 if (isNaN(a) || isNaN(b)) {a='';b='';total='' }
  else {
   total=a+b
  }
 acc=Request.ServerVariables('HTTP_ACCEPT')+''
 if (acc.indexOf('message/x-jl-formresult')!=-1) {
  Response.Write(total)
 } else {
%>
<script src="xmlhttp.js" type="text/javascript"></script>
<script>
 function calc() {
  frm=document.forms[0]
  url="add.1?a="+frm.elements['a'].value+"&b="+frm.elements['b'].value
  xmlhttp.open("GET",url,true);
  xmlhttp.onreadystatechange=function() {
   if (xmlhttp.readyState==4) {
    document.forms[0].elements['total'].value=xmlhttp.responseText
   }
  }
 xmlhttp.setRequestHeader('Accept','message/x-jl-formresult')
 xmlhttp.send()
 return false
}
</script>
<form action="add.1" method="get" onsubmit="return calc()">
<input type=text name=a value="<%=a%>"> + <input type=text name=b value="<%=b%>">
 = <input type=text name=total value="<%=total%>">
<input type=submit value="Calculate">
</form>
<%
}
%>
```

Try the example page

The example above uses JScript in ASP as the server side language, the HTTP ACCEPT header is used to tell the server which response to send back - either the full page or just the result. The HTTP ACCEPT header is used to tell servers what mime-types the client will accept, normally it says things like text/html etc. Here though we tell it we only accept "message/x-jl-formresult", so the server knows it is our client (or another client, who knows about "message/x-jl-formresult") making the request.

Other methods of identifying what to return may be appropriate depending on the type of data you send to the server, or you could simply use different urls for the form submission and xmlhttp request, whatever you do, remember to have sensible fallback to the non-xml http request browsers where possible.

Using JSON as the transfer language

Whilst XML can be used to encode the information you retrieve with the object and it will be available in the responseXML property, however xml is less well supported, some browsers require that the content type of the resource is one of only 2 possible XML mime-types text/xml or application/xml for the property to be populated, and there are always the normal well formness problems you always get with XML. JSON

is a good alternative, it's fast to parse, and much, much faster to access in script.

I use JSON in the Flight Routeplanner to look up information on airports, an Example with London Heathrow, you can easily parse the returned JSON into a script object using the new Function constructor, it checks the status as the script returns 404 if it fails to find an airport with that iata code.

```
xmlhttp.open("GET","/routeplanner/airport.1?LHR",true);
xmlhttp.onreadystatechange=function() {
 if (xmlhttp.readyState==4) {
  if (xmlhttp.status!=404) {
   var local=new Function("return "+xmlhttp.responseText)();
   alert("Code - Name\n"+local[0].id+' - '+local[0].name);
  } else {
   alert("Airport not found");
  }
 }
}
xmlhttp.send(null);
```

Try the example.

Using XMLHTTP with GOOGLE's SOAP API

Google provides a SOAP interface to it's database. You need to register for a key that lets you make 1000 a day, to make a request. You then need to parse the returned XML.

```
search="Word"
xmlhttp.open("POST", "http://api.google.com/search/beta2",true);
xmlhttp.onreadystatechange=function() {
 if (xmlhttp.readyState==4) {
  alert(xmlhttp.responseText)
 }
}
xmlhttp.setRequestHeader("Man", "POST http://api.google.com/search/beta2 HTTP/1.1")
xmlhttp.setRequestHeader("MessageType", "CALL")
xmlhttp.setRequestHeader("Content-Type", "text/xml")

xmlhttp.send("<?xml version='1.0' encoding='UTF-8'?>"+"\n\n"+"<SOAP-ENV:Envelope"+
     ' xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"'+
     ' xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"'+
     ' xmlns:xsd="http://www.w3.org/1999/XMLSchema">'+
     '<SOAP-ENV:Body><ns1:doGoogleSearch'+
     ' xmlns:ns1="urn:GoogleSearch"'+
     ' SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">'+
     '<key xsi:type="xsd:string">GOOGLEKEY</key> <q'+
     ' xsi:type="xsd:string">'+search+'</q> <start'+
     ' xsi:type="xsd:int">0</start> <maxResults'+
     ' xsi:type="xsd:int">10</maxResults> <filter'+
     ' xsi:type="xsd:boolean">true</filter> <restrict'+
     ' xsi:type="xsd:string"></restrict> <safeSearch'+
     ' xsi:type="xsd:boolean">false</safeSearch> <lr'+
     ' xsi:type="xsd:string"></lr> <ie'+
     ' xsi:type="xsd:string">latin1</ie> <oe'+
     ' xsi:type="xsd:string">latin1</oe>'+
     '</ns1:doGoogleSearch>'+
   '</SOAP-ENV:Body></SOAP-ENV:Envelope>')
```

Google is using a SOAP interface, many people think SOAP has some serious issues worth considering. REST is probably a better model as it works with the current web framework, proxies, caches etc. So whilst we can use the XML HTTP Request object to talk soap, it's probably best not to unless you have no

control over what's happening on the server end. (Thanks to Dan Schmierer for pointing out an error in my script.)

By default the object can only call back to the same server, in a reduced security environment (accessed from file:// say) IE can access any domain, Mozilla can also do that if you request and are granted the appropriate permissions see "a google thread I can't get to offline!"

Nearby...

- Dynamically updating SVG.

[1] Actually a lot of the "AJAX" applications make little use of this object, using the older and often more flexible IFRAME remote scripting methods, but they could've been using this object, and AJAX should be thought of more as a concept of the kind of applications that can be created with the object.

Jim Ley - jim@jibbering.com, Jibbering.com