Ajax
*A*synchronous *Ja*vaScript + *X*ML

Mark Andreessen, Netscape, 1995: "MS Windows will be reduced to a poorly debugged set of device drivers running under Netscape Navigator, with desktop-style applications running inside the browser". This did not happen until 10 years later (true/false?)

Asynchronous *Ja*vaScript + *X*ML

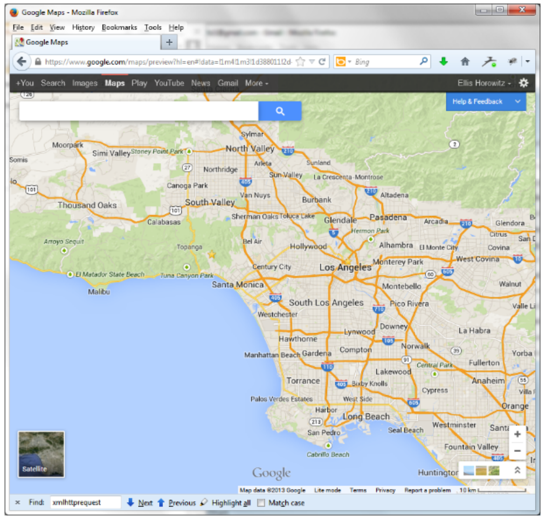
- Ajax isn’t a technology.
- It’s really several technologies. Ajax incorporates:
  - standards-based presentation using XHTML;
  - CSS, dynamically manipulated using JavaScript;
  - dynamic display and interaction using the Document Object Model (DOM). Web page exposed as DOM object;
  - data interchange using XML and data manipulation using XSLT (not supported by all browsers);
  - asynchronous data retrieval using XMLHttpRequest, a JavaScript object, a.k.a “Web remoting”;
  - JavaScript binding everything together;
  - Server no longer performs display logic, only business logic.
- Acronym originated by Jesse James Garrett

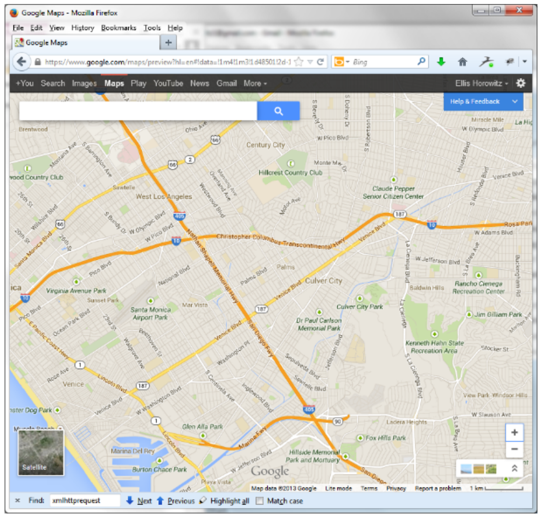
Some History and Browsers Supporting Ajax

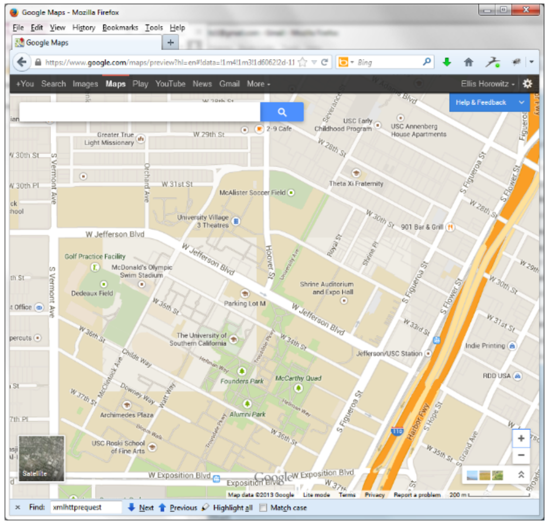
- The XMLHttpRequest object is the main element of Ajax programming
- Microsoft first implemented the XMLHttpRequest object in Internet Explorer 5 for Windows as an ActiveX object.
- Similar functionality is covered in a recommended W3C standard, Document Object Model (DOM) Level 3 Load and Save Specification (April 2004):

  http://www.w3.org/TR/DOM-Level-3-LS
- Engineers on the Mozilla project implemented a compatible native version for Mozilla 1.0 (included in Netscape 7, Firefox 1.0 and later releases). Apple has done the same starting with Safari 1.2.
- Other browsers supporting XMLHttpRequest include:
  - Opera 7.6+, Apple Safari 1.2+, All Mobile browsers
- XMLHttpRequest is being standardized by the W3C. Working Draft for Level 1 available at:
  - http://www.w3.org/TR/XMLHttpRequest/

An Example Using Ajax - Google Maps







Initial screen

zoom 3 times

drag map east and zoom

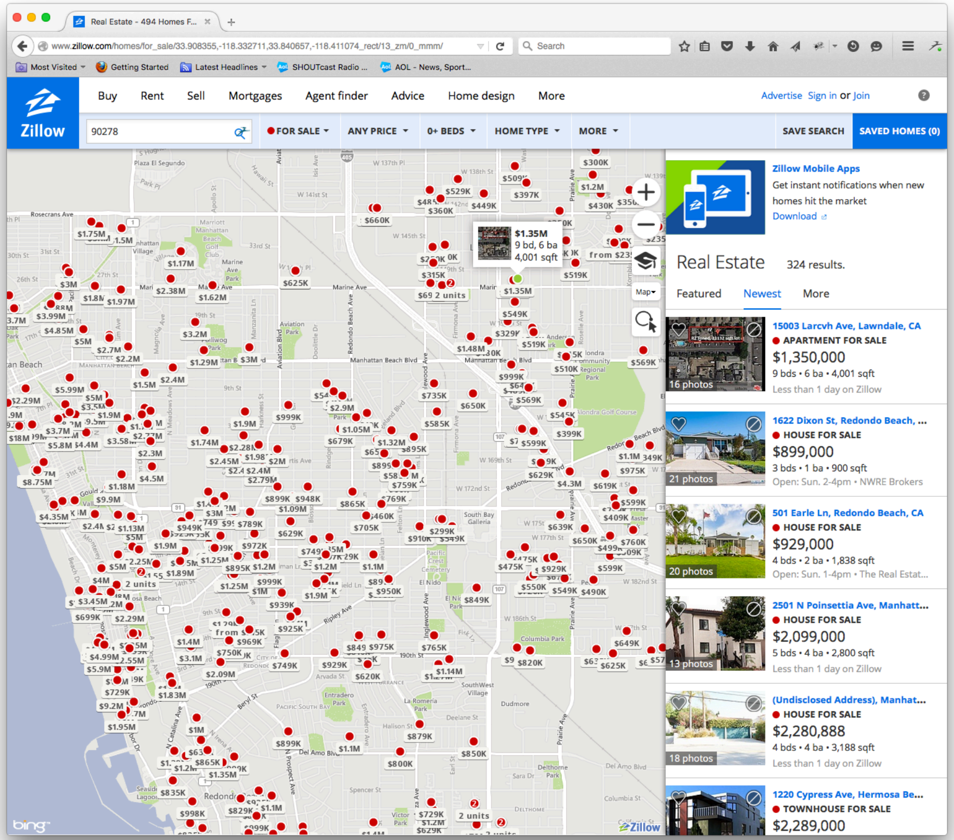
Notice that the page is never explicitly refreshed
View source and search for XMLHttpRequest;
You will find multiple occurrences

A Mash-Up Combines Multiple Sources of Data

A “mash-up” is a web application that consumes ("remixes") content from different sources and aggregates them to create a new application

Mashup Example – www.zillow.com

A combination of satellite photos with records of home sale prices placed on top of the appropriate houses



Zillow Mobile Apps
Get instant notifications when new homes hit the market
Download

Real Estate 324 results.

Featured Newest More

15003 Larcvh Ave, Lawndale, CA
APARTMENT FOR SALE
$1,350,000
9 bds • 6 ba • 4,001 sqft
Less than 1 day on Zillow
16 photos

1622 Dixon St, Redondo Beach, ...
HOUSE FOR SALE
$899,000
3 bds • 1 ba • 900 sqft
Open: Sun. 2-4pm • NWRE Brokers
21 photos

501 Earle Ln, Redondo Beach, CA
HOUSE FOR SALE
$929,000
4 bds • 2 ba • 1,838 sqft
Open: Sun. 1-4pm • The Real Estat...
20 photos

2501 N Poinsettia Ave, Manhatt...
HOUSE FOR SALE
$2,099,000
5 bds • 4 ba • 2,800 sqft
Less than 1 day on Zillow
13 photos

(Undisclosed Address), Manhat...
HOUSE FOR SALE
$2,280,888
4 bds • 4 ba • 3,188 sqft
Less than 1 day on Zillow
18 photos

1220 Cypress Ave, Hermosa Be...
TOWNHOUSE FOR SALE
$2,289,000

Characteristics of Ajax Applications

- They are applications, not just web sites
- They allow for smooth, continuous interaction
- "Live" content
- Visual Effects
- Animations, dynamic icons
- Single keystrokes can lead to server calls
- New Widgets (selectors, buttons, tabs, lists)
- New Styles of Interaction (drag-and-drop, keyboard shortcuts, double-click)

Comparing Traditional vs. AJAX Websites

Traditional

- Interface construction is mainly the responsibility of the server
- User interaction is via form submissions
- An entire page is required for each interaction (bandwidth)
- Application is unavailable while an interaction is processing (application speed)

Ajax

- Interface is manipulated by client-side JavaScript manipulations of the Document Object Model (DOM)
- User interaction via HTTP requests occur 'behind the scenes'
- Communication can be restricted to data only
- Application is always responsive

How to Recognize an Ajax Application Internally

“View Source” in the browser and search for:

- Javascript code that invokes any of these APIs:
  - XMLHttpRequest or ActiveXObject("Microsoft.XMLHTTP")
- JavaScript that “loads” other JavaScript code (files with .js extension)
- XML code passed as text strings to a server, such as ‘<?xml version="1.0"><page>…</page>’
- Javascript <script> sections that embed code between //<![CDATA[and //]]>
- JavaScript code that creates IFRAMEs, such as window.document.createElement("iframe")

The Classic Web Application Model

- Most user actions in the browser interface trigger an HTTP request back to a web server.
- The server does some processing — retrieving data, crunching numbers, talking to various legacy systems.
- The server then returns an HTML page to the client.

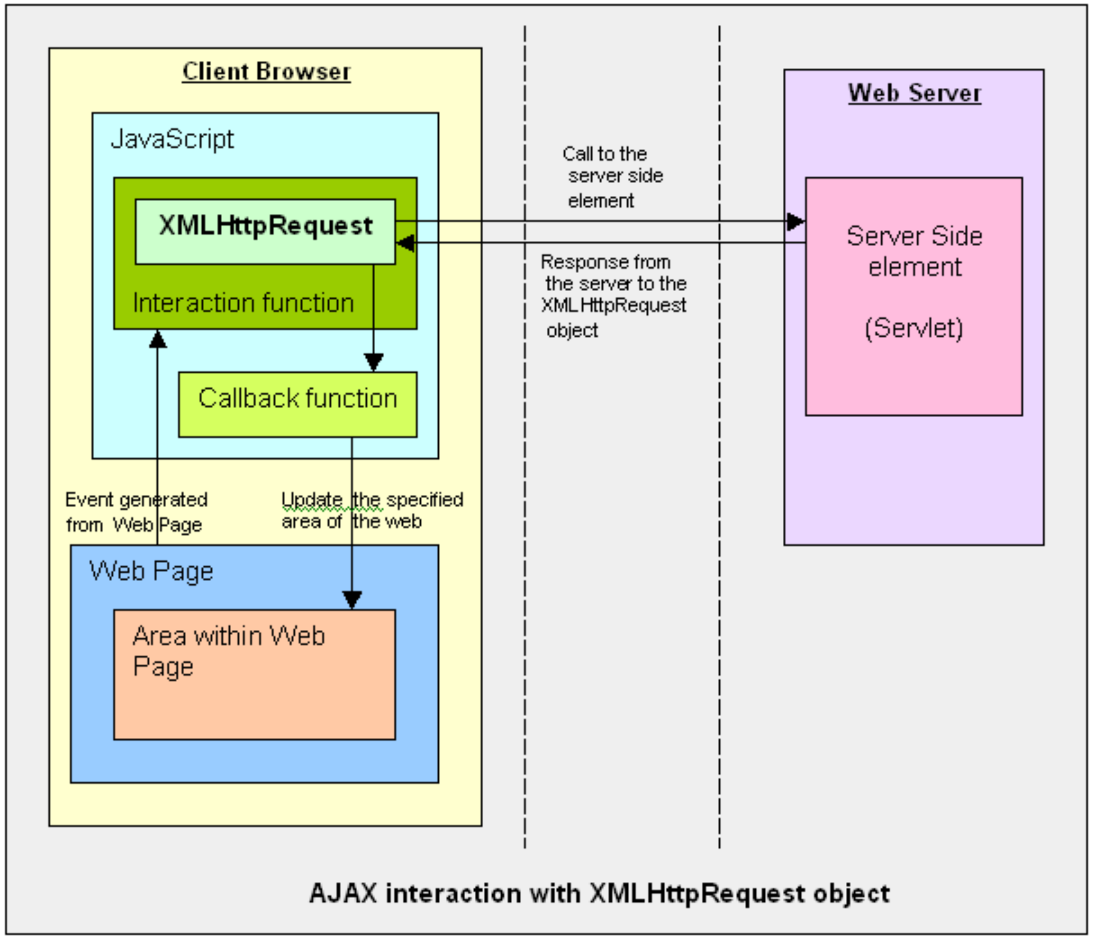
- Approach issues:
  - It doesn’t make for a great user experience.
  - While the server is doing its thing, the user is waiting.
  - And at every step in a task, the user waits some more.

The Ajax Web Application Model

- Ajax introduces an intermediary — an Ajax engine — between the user and the server.
- Instead of loading a webpage, at the start of the session, the browser loads an Ajax engine — written in JavaScript and usually stored in a hidden frame.
- This engine is responsible for
  - rendering the interface the user sees
  - communicating with the server on the user's behalf.
- The Ajax engine allows the user's interaction with the application to happen asynchronously — independent of communication with the server.
- Approach Benefits:
  - An Ajax application eliminates the start-stop-start-stop nature of interaction on the Web.
  - The user is never staring at a browser window with hourglass, waiting for the server to do something.
  - The application is more responsive.

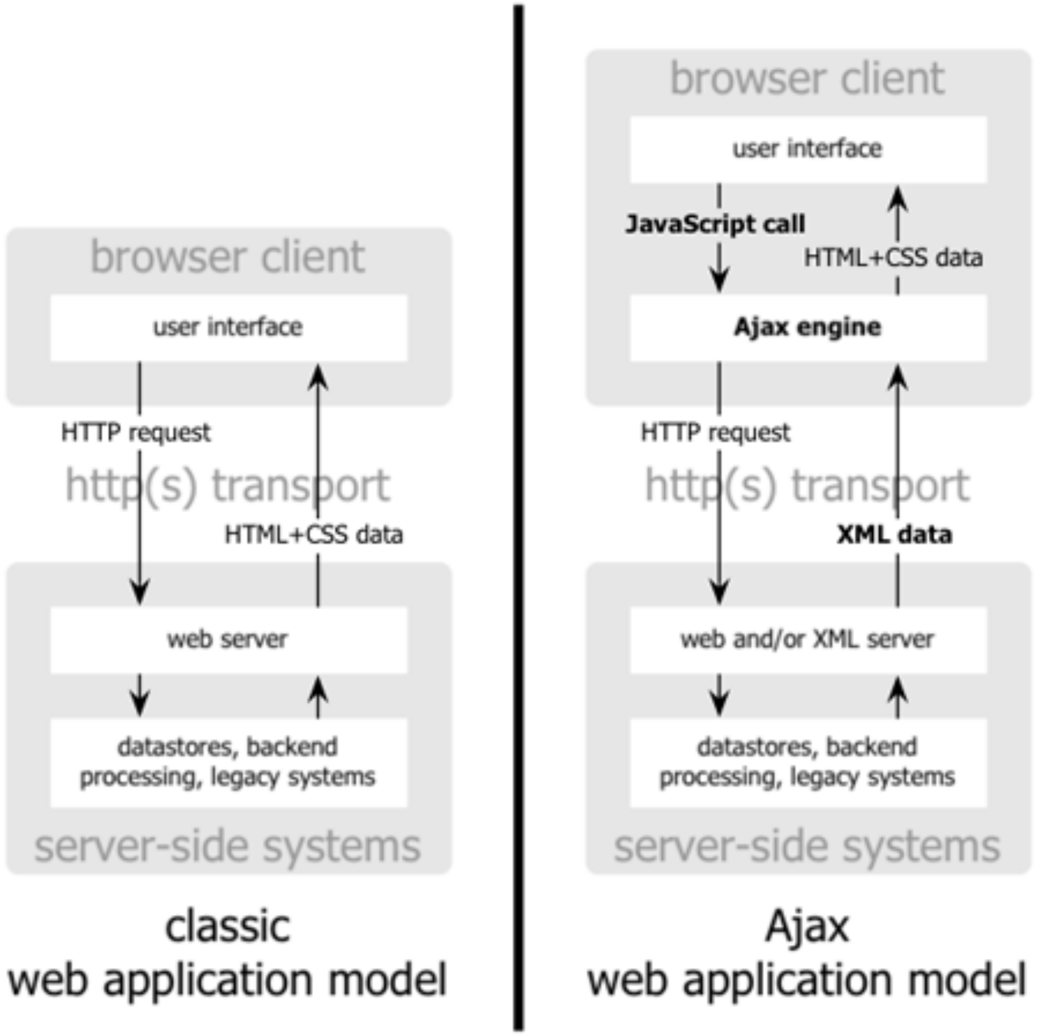
AJAX:

- Cuts down on user wait time
- Uses client to offload some work from the server
- Asynchronous operation

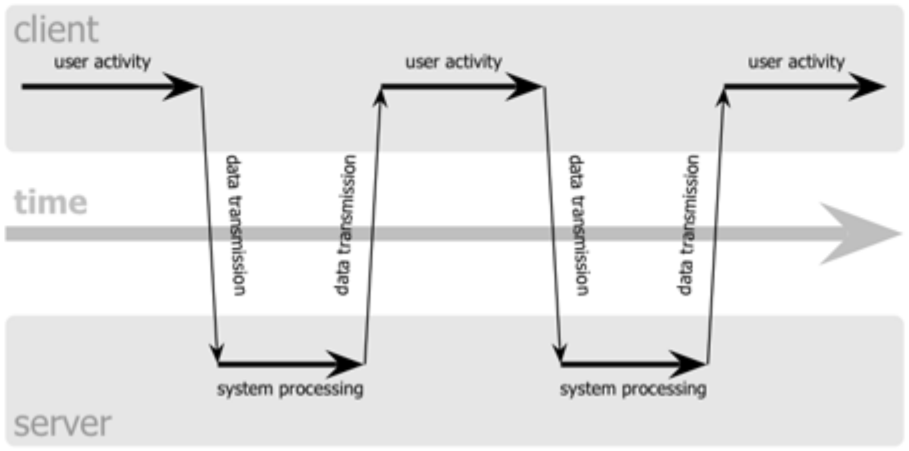


Client Browser
JavaScript
XMLHttpRequest
Interaction function
Callback function
Event generated from Web Page
Update the specified area of the web
Web Page
Area within Web Page
Call to the server side element
Response from the server to the XMLHttpRequest object
Web Server
Server Side element
(Servlet)
AJAX interaction with XMLHttpRequest object

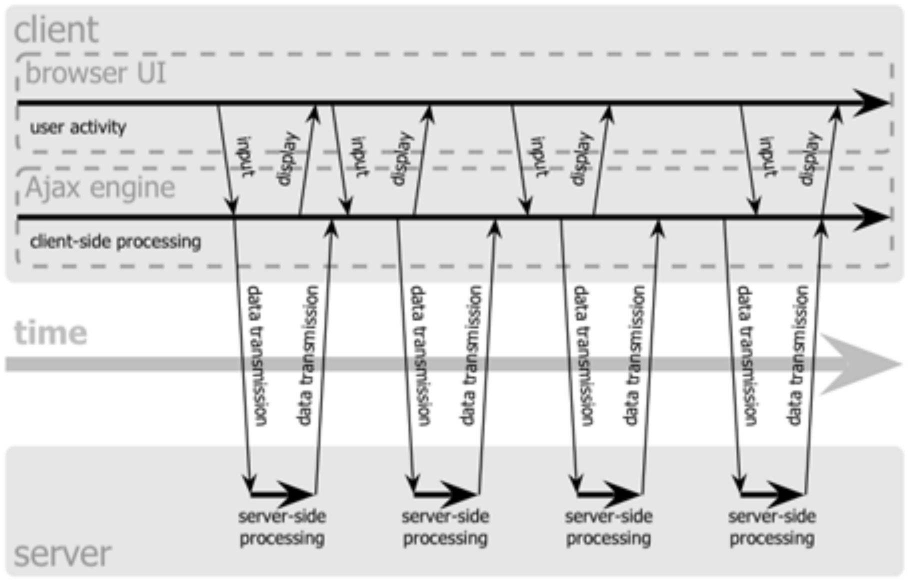
Traditional Web Applications Model compared to the Ajax Model



Classic Web Application Model (synchronous)



Ajax Web Application Model (asynchronous)



Ajax Engine Role

- Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead.
- Any response to a user action that doesn't require a trip back to the server, such as:
  - simple data validation
  - editing data in memory
  - even some navigation

  the engine handles on its own.
- If the engine needs something from the server in order to respond, such as:
  - submitting data for processing
  - loading additional interface code
  - retrieving new data

  the engine makes those requests asynchronously, usually using XML, without stalling a user's interaction with the application.

Initiating the XMLHttpRequest Object

- Creating an instance of the XMLHttpRequest object requires branching syntax to account for browser differences. For Safari, Chrome, FF, IE 9+, Edge a simple call to the object's constructor function does the job:

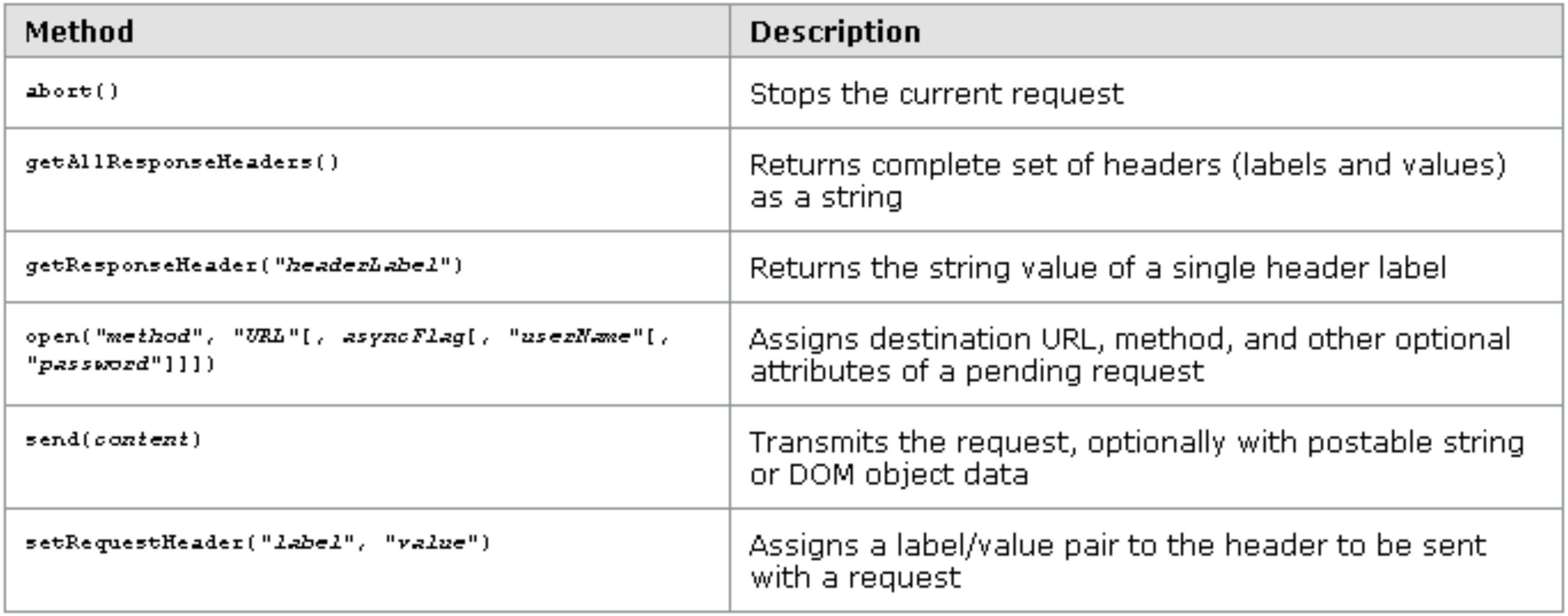
  ```
  var req = new XMLHttpRequest();
  ```

- For Internet Explorer 5-8, pass the name of the object to the ActiveX constructor:

  ```
  var req = new ActiveXObject("Microsoft.XMLHTTP");
  ```

- The object reference returned by both constructors is to an abstract object that works entirely out of view of the user. Its methods control all operations, while its properties hold, among other things, various data pieces returned from the server.

XMLHttpRequest Object Methods



Method	Description
`abort()`	Stops the current request
`getAllResponseHeaders()`	Returns complete set of headers (labels and values) as a string
`getResponseHeader("`*headerLabel*`")`	Returns the string value of a single header label
`open("`*method*`", "`*URL*`"[, `*asyncFlag*`[, "`*userName*`"[, "`*password*`"]]])`	Assigns destination URL, method, and other optional attributes of a pending request
`send(`*content*`)`	Transmits the request, optionally with postable string or DOM object data
`setRequestHeader("`*label*`", "`*value*`")`	Assigns a label/value pair to the header to be sent with a request

XMLHttpRequest Object Methods (cont'd)

- Of the methods shown in the Table on the previous slide, the **open()** and **send()** methods are the ones you'll likely use most.
- **open()** sets the scene for an upcoming operation. Two required parameters are the HTTP method you intend for the request and the URL for the connection. For the method parameter, use "GET" on operations that are primarily data retrieval requests; use "POST" on operations that send data to the server, especially if the length of the outgoing data is potentially greater than 512 bytes. The URL may be either a complete or relative URL.
- It is safer to **send** asynchronously and design your code around the onreadystatechange event for the request object.

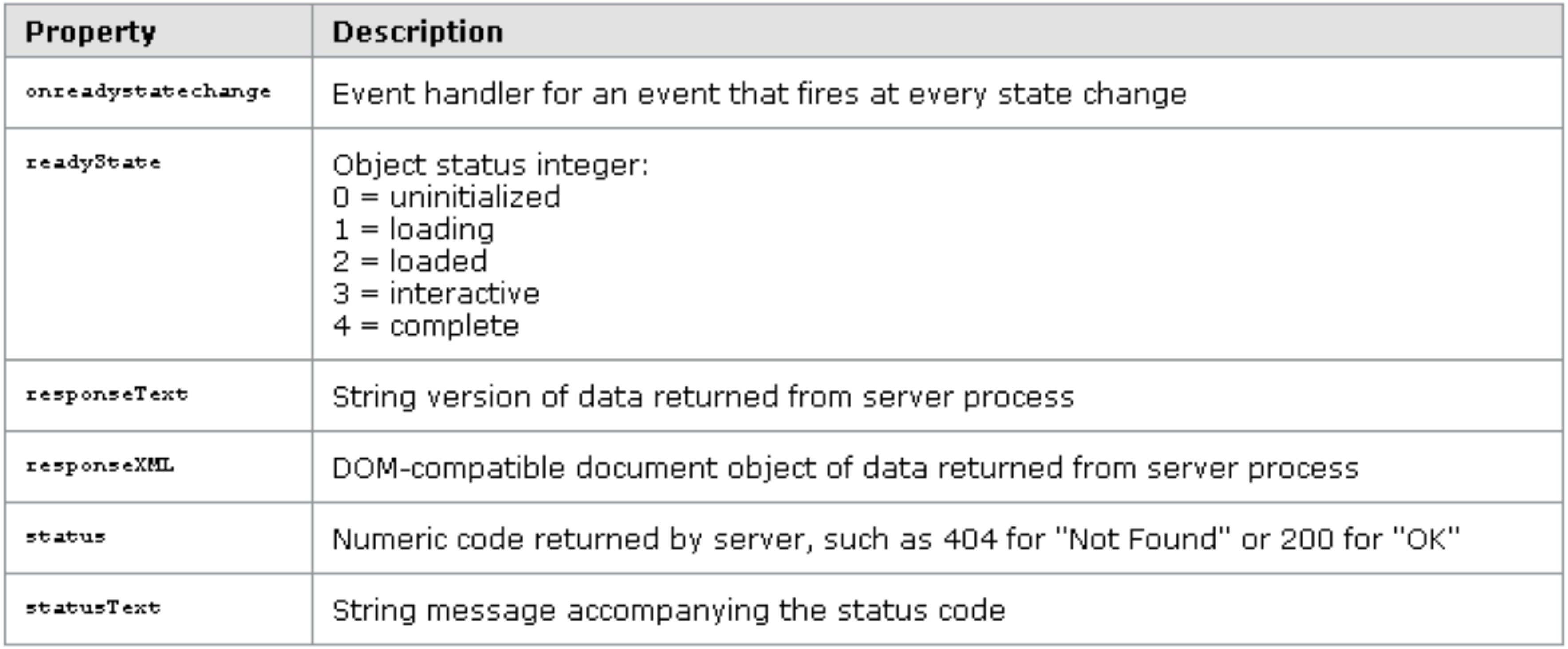
XMLHttpRequest Example Code

```
var req;

function loadXMLDoc(url) {
    req = false;
    // branch for native XMLHttpRequest object
    if(window.XMLHttpRequest) {
          try {   req = new XMLHttpRequest();
        } catch(e) {  req = false;
        }
    // branch for IE/Windows ActiveX version
    } else if(window.ActiveXObject) {
          try {
          req = new ActiveXObject("Msxml2.XMLHTTP");
          } catch(e) {
          try {  req = new ActiveXObject("Microsoft.XMLHTTP");
          } catch(e) {   req = false;
          }
          }
    }
    if(req) {
          req.onreadystatechange = processReqChange;
          req.open("GET", url, true);
          req.send("");
    }
}
```

This code instantiates an XmlHttpRequest object depending upon the browser

XMLHttpRequest Object Properties



Property	Description
onreadystatechange	Event handler for an event that fires at every state change
readyState	Object status integer: 0 = uninitialized 1 = loading 2 = loaded 3 = interactive 4 = complete
responseText	String version of data returned from server process
responseXML	DOM-compatible document object of data returned from server process
status	Numeric code returned by server, such as 404 for "Not Found" or 200 for "OK"
statusText	String message accompanying the status code

XMLHttpRequest Object Properties (cont'd)

- Use the **readyState** property inside the event handler function that processes request object state change events. While the object may undergo interim state changes during its creation and processing, the value that signals the completion of the transaction is 4.
- Access data returned from the server via the **responseText** or **responseXML** properties. The former provides only a string representation of the data. More powerful, however, is the XML document object in the responseXML property. This object is a full-fledged document node object (a DOM nodeType of 9), which can be examined and parsed using W3C Document Object Model (DOM) node tree methods and properties.
- Note, however, that this is an XML, rather than HTML, document, meaning that you cannot count on the DOM's HTML module methods and properties. This is not really a restriction because the Core DOM module gives you ample ways of finding element nodes, element attribute values, and text nodes nested inside elements.

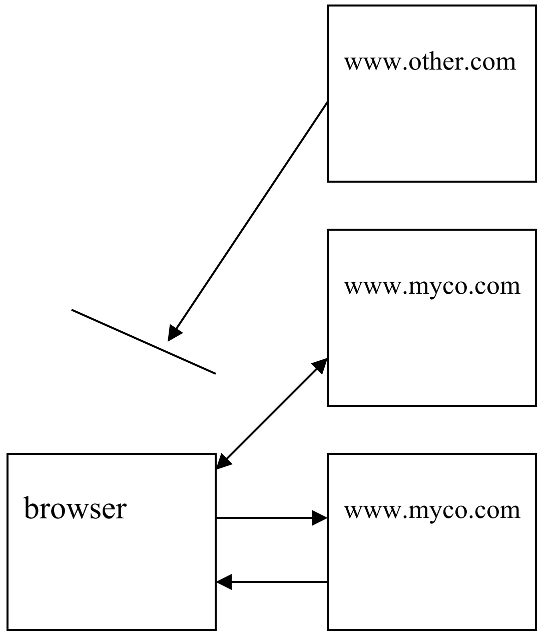
onreadystatechange Event Handler Function

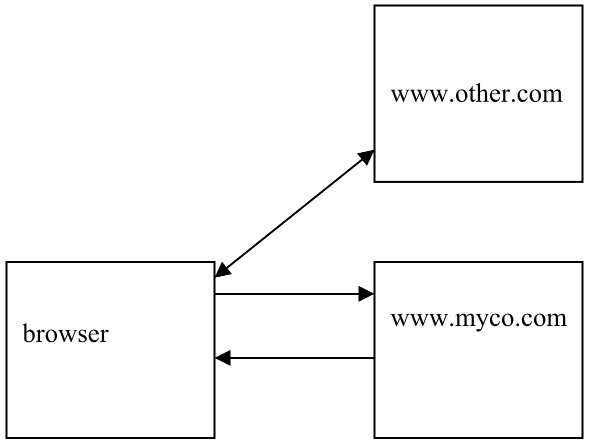
```
function processReqChange() {
    // only if req shows "loaded"
    if (req.readyState == 4) {
        // only if "OK"
        if (req.status == 200) {
             // processing statements req.responseText
             // for JSON
             // and req.responseXML for XML go here...
        } else {
alert("There was a problem retrieving the XML data:\n"
   +   req.statusText);
        }
    }
}
```

Security Issues

- When the XMLHttpRequest object operates within a browser, it **adopts the same-domain security policies** of typical JavaScript activity (sharing the same "sandbox," as it were).
- First, on most browsers supporting this functionality, the page that bears scripts accessing the object needs to be retrieved via http: protocol, meaning that **you won't be able to test the pages from a local hard disk** (file: protocol) without some extra security issues cropping up, especially in Mozilla and IE on Windows.
- Second, the domain of the URL request destination must be the same as the one that serves up the page containing the script. This means, unfortunately, that client-side scripts cannot fetch web service data from other sources, and blend that data into a page. **Everything must come from the same domain.**

AJAX Cross Domain Security





For security reasons, scripts are only allowed to access data which comes from the same domain

The one exception is for images: images can come from any domain, without any security risk.

This is why all the mash-up applications involve images

They simply would not be possible for other kinds of data

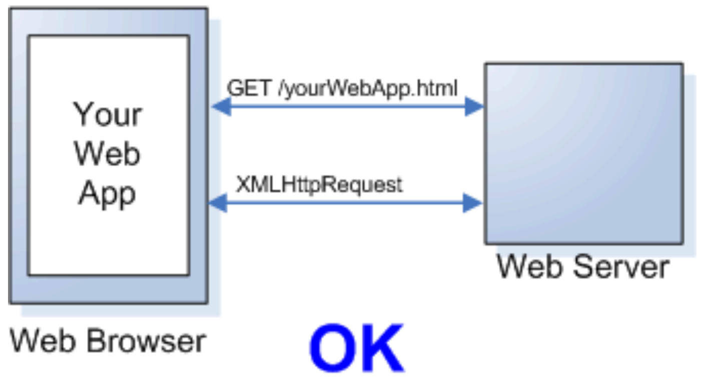
Cross-domain Restrictions and A Solution

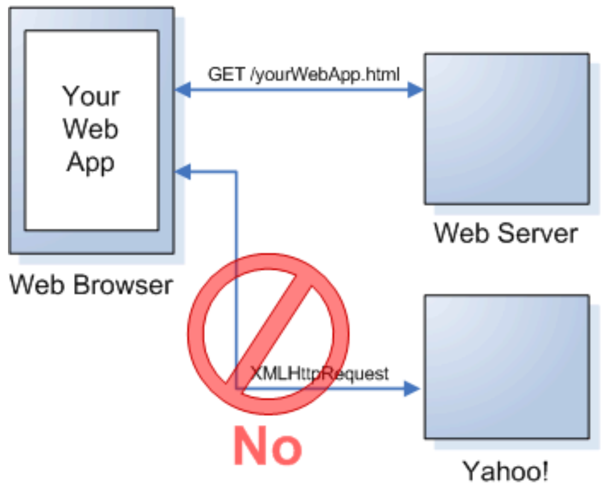
- Browser security restrictions prevent your web application from opening network connections to domains other than the one your application came from.
- For example, suppose your web application wants to use data both from your site and from Yahoo!; normally this is not possible as it is a violation of browser cross-domain security policy.
- **One way** to work around this issue is to install a web proxy on your server that will pass requests from your application to Yahoo! and the data back again.
- If you are using a proxy to relay requests from your web application to Yahoo!, the actual request URL you use from your web application is different, as you must relay your request through your web server proxy.
- For example, if you are using the PHP Proxy, the actual request looks something like this:

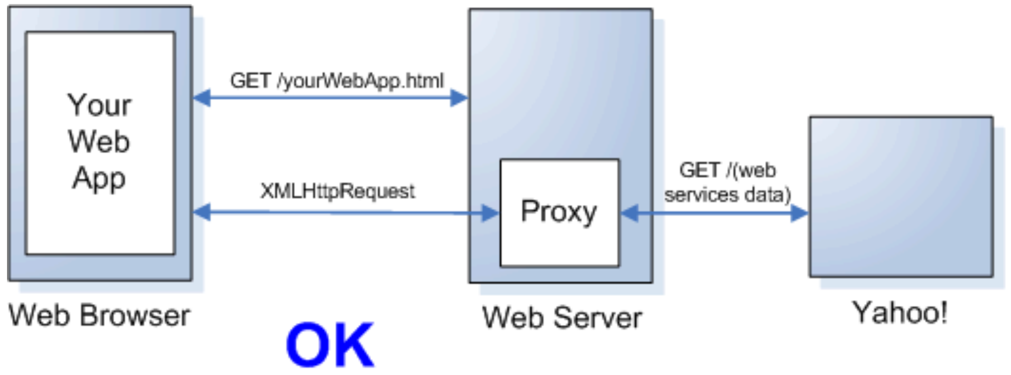
http://www.*yourdomain*.com/php_proxy_simple.php?WebSearchService/V1/webSearch?appid=YahooDemo&query=persimmon&results=2

where the latter part of the URL is used to access the data on Yahoo!

Why You Need a Proxy



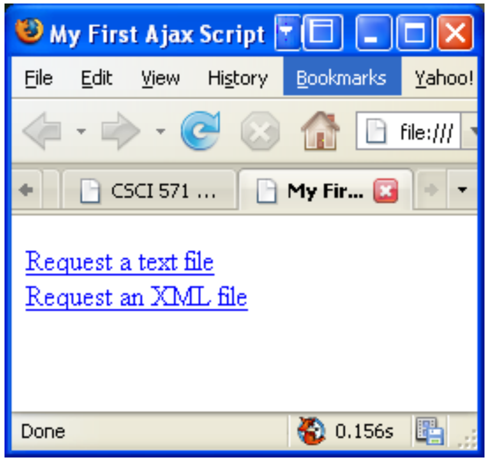




A First Ajax Example – Using Ajax to Download Files

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN">
<html><head>
   <title>First Ajax Script</title>
   <script src="script01.js" type="text/javascript"
   language="Javascript">
   </script>
</head><body>
   <p><a id="makeTextRequest" href="gAddress.txt">Request a
   text file</a><br />
   <a id="makeXMLRequest" href="us-states.xml">
Request an XML file</a></p>
   <div id="updateArea">&nbsp;</div>
</body>
</html>
```

The javascript file does all of the work



Request a text file
Request an XML file

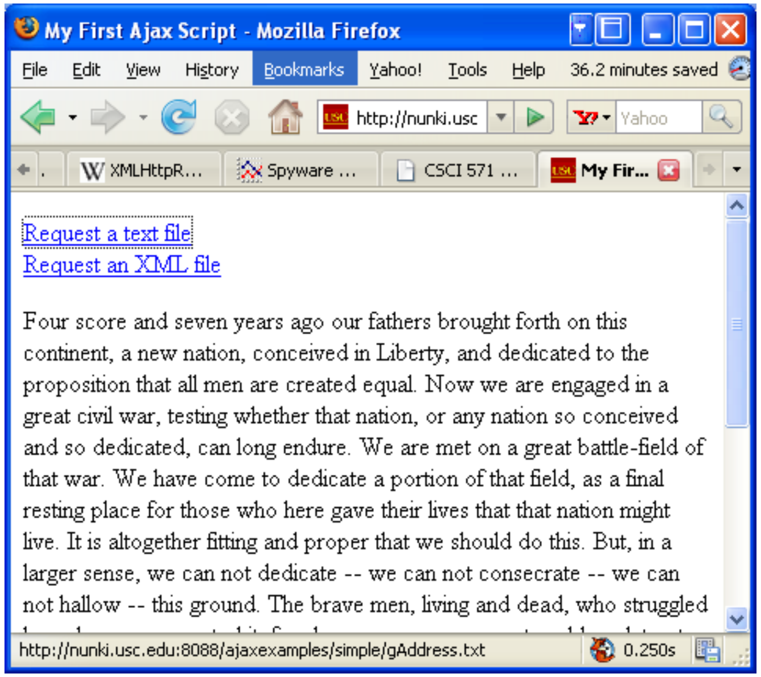
Imported JavaScript-script01.js

```
window.onload = initAll;
var xhr = false;
function initAll() {
    document.getElementById("makeTextRequest").onclick = getNewFile;
    document.getElementById("makeXMLRequest").onclick = getNewFile;}
function getNewFile() {
    makeRequest(this.href);  return false;}
function makeRequest(url) {
    if (window.XMLHttpRequest) {  xhr = new XMLHttpRequest();}
    else { if (window.ActiveXObject) {
                try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }
                catch (e) { }
          }   }
    if (xhr) {  xhr.onreadystatechange = showContents;
          xhr.open("GET", url, true);  xhr.send(null);  }
    else { document.getElementById("updateArea").innerHTML = "Sorry, but I couldn't
    create an XMLHttpRequest";  }  }
function showContents() {
    if (xhr.readyState == 4) {
          if (xhr.status == 200) {
                var outMsg = (xhr.responseXML &&
    xhr.responseXML.contentType=="text/xml") ?
    xhr.responseXML.getElementsByTagName("choices")[0].textContent : xhr.responseText;
    } else { var outMsg = "There was a problem with the request " + xhr.status; }
          document.getElementById("updateArea").innerHTML = outMsg;   } }
```

On page load the onclick event is set to call the function
When the click is made, getNewFile and makerequest are executed.

showContents waits for a successful return of an file; it then prints the result in the browser

Browser Output

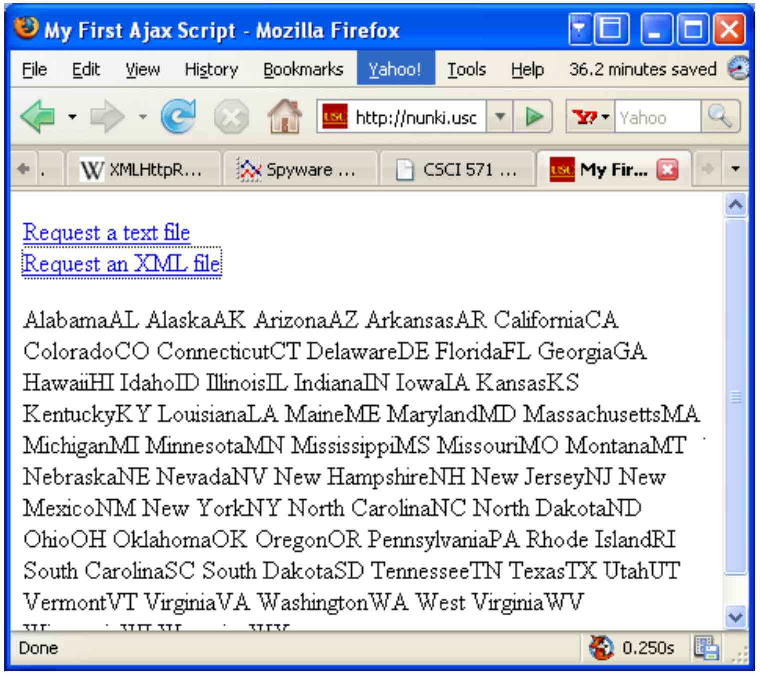


My First Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help 36.2 minutes saved
http://nunki.usc
Yahoo
XMLHttpR... Spyware ... CSCI 571 ... My Fir...

Request a text file
Request an XML file

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this ground. The brave men, living and dead, who struggled

http://nunki.usc.edu:8088/ajaxexamples/simple/gAddress.txt 0.250s



My First Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help 36.2 minutes saved
http://nunki.usc
Yahoo
XMLHttpR... Spyware ... CSCI 571 ... My Fir...

Request a text file
Request an XML file

AlabamaAL AlaskaAK ArizonaAZ ArkansasAR CaliforniaCA ColoradoCO ConnecticutCT DelawareDE FloridaFL GeorgiaGA HawaiiHI IdahoID IllinoisIL IndianaIN IowaIA KansasKS KentuckyKY LouisianaLA MaineME MarylandMD MassachusettsMA MichiganMI MinnesotaMN MississippiMS MissouriMO MontanaMT NebraskaNE NevadaNV New HampshireNH New JerseyNJ New MexicoNM New YorkNY North CarolinaNC North DakotaND OhioOH OklahomaOK OregonOR PennsylvaniaPA Rhode IslandRI South CarolinaSC South DakotaSD TennesseeTN TexasTX UtahUT VermontVT VirginiaVA WashingtonWA West VirginiaWV

Done 0.250s

Result of clicking on the first link

Result of clicking on the second link

http://cs-server.usc.edu:45678/ajaxexamples/simple/script01.html

Second Ajax Example – Using Ajax to Download Files from Flickr

- **Here is the html file, which basically loads script02.js**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html><head><title>Second Ajax Script</title>
<script src="script02.js" type="text/javascript" language="Javascript"></script>
</head><body><div id="pictureBar"> </div></body></html>
```

- **Here is script02.js**

```
window.onload = initAll;
var xhr = false;
function initAll() {
    if (window.XMLHttpRequest) { xhr = new XMLHttpRequest(); }
    else { if (window.ActiveXObject) {
          try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) { }  } }
    if (xhr) { xhr.onreadystatechange = showPictures;
          xhr.open("GET", "flickrfeed.xml", true);   xhr.send(null);  }
    else {  alert("Sorry, but I couldn't create an XMLHttpRequest");  }  }
function showPictures() {
    var tempDiv = document.createElement("div");
    var pageDiv = document.getElementById("pictureBar");
    if (xhr.readyState == 4) {
          if (xhr.status == 200) {
                    tempDiv.innerHTML = xhr.responseText;
                    var allLinks = tempDiv.getElementsByTagName("a");
                              for (var i=1; i<allLinks.length; i+=2) {
                    pageDiv.appendChild(allLinks[i].cloneNode(true)); }  }
    else { alert("There was a problem with the request " + xhr.status); }  }  }
```

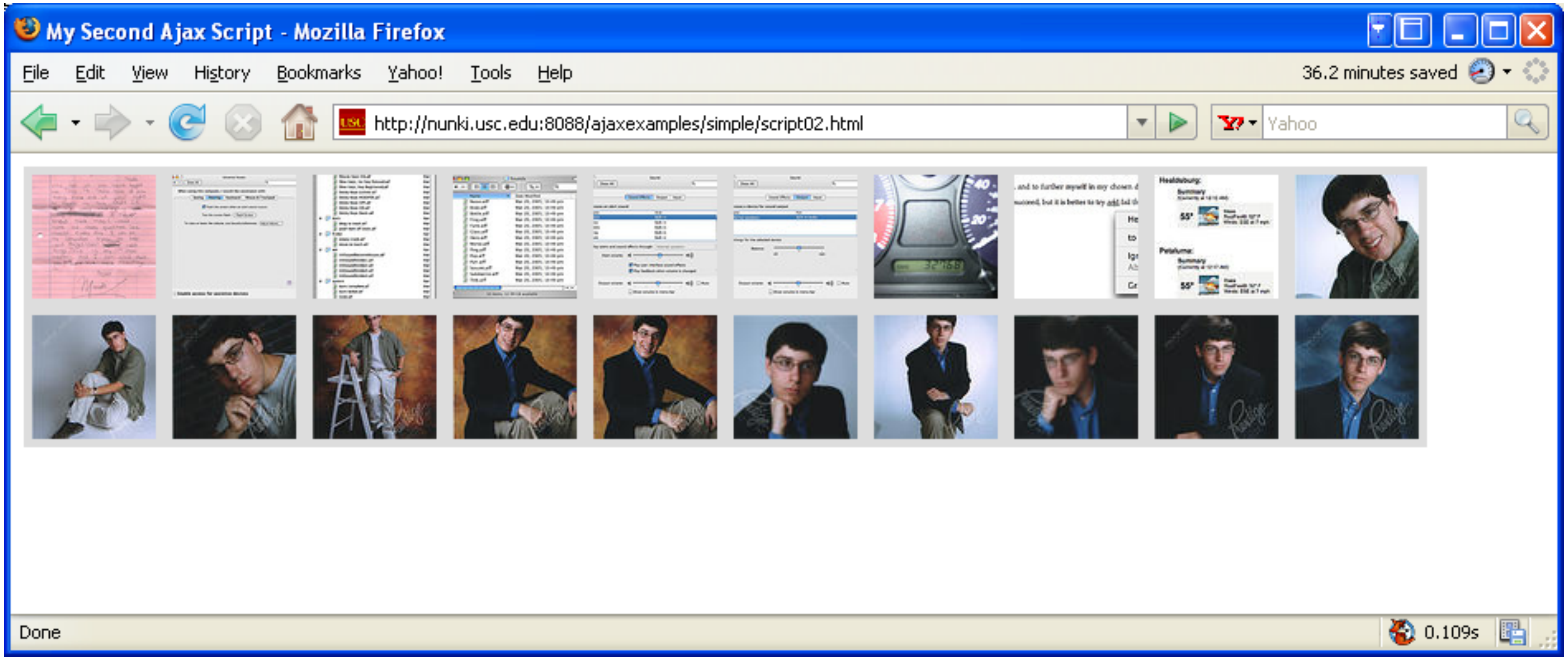
ShowPictures retrieves an file from flickr;
The result is extracted from responseText and assigned to innerHTML property

Portion of Flickr XML file

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xmlns=http://www.w3.org/2005/Atom
Xmlns=http://purl.org/dc/elements/1.1>
<title>Dori Simth's Photos</title>
<link rel="self"
    href=http://www.flickr.com/services/feeds/photos_public.gne?id=23922109@N00 />
<link rel="alternate" type="text/html" href=http://www.flickr.com/photos/dorismith/
    />
<id>tag:flickr.com,2005:/photos/public/116078</id>
<icon>http://static.flickr.com/5/buddyicons/23922109@N00.jpg?1113973282</icon>
<subtitle>A feed of Dori Smith's Photos</subtitle>
<updated>2006-03-22T20:12:44Z</updated>
<generator uri=http://www.flickr.com/>Flickr</generator>
<entry>
<title>Mash note</title>
<link rel="alternate" type="text/html"
    href=http://www.flickr.com/photos/dorismith/116463569/ />
OTHER STUFF
<p> <a href=http://www.flickr.com/photos/dorismith/116463569/ title="Mash
    note"><img src=http://static.flickr.com/44/116463569_483fd4ee7c_s.jpg
    width="75" height="75" alt="Mash note" style="border: 5px solid #ddd;" /></a>
    </p>
```

Each <entry> node has two links;
This application uses the second link so the showPictures loop starts with 1 rather than 0 and increments by 2; each link contains the thumbnail image inside it; every thumbnail is a link back to the original photo

Browser Output


My Second Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
36.2 minutes saved
http://nunki.usc.edu:8088/ajaxexamples/simple/script02.html
Yahoo
Done
0.109s

http://cs-server.usc.edu:45678/ajaxexamples/simple/script02.html

Third Ajax Example - Refreshing Server Data

- This extension retrieves a new version of the data from the server, refreshing the page; **here is the html accessing javascript**

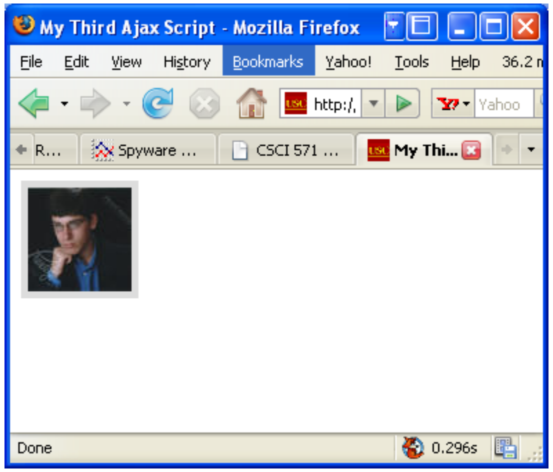
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html><head><title>My Third Ajax Script</title>
<script src="script03.js" type="text/javascript" language="Javascript"></script></head>
<body><div id="pictureBar"> </div></body></html>
```

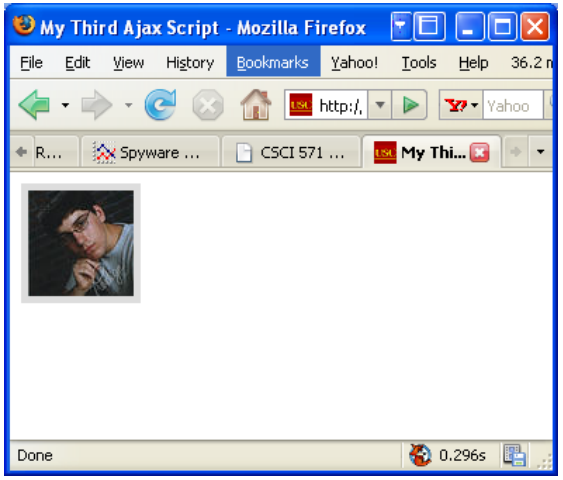
- **And here is the source for script03.js**

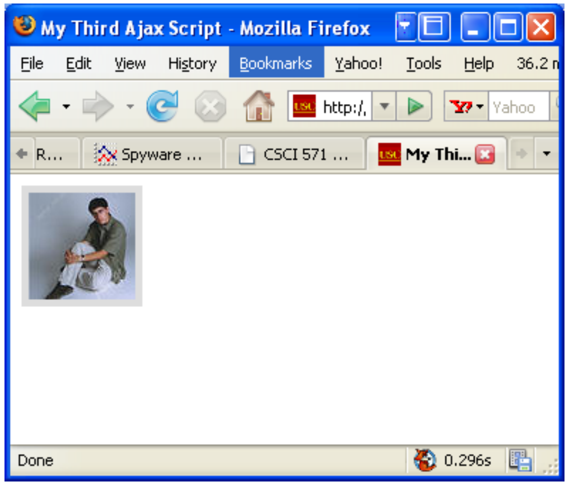
```
window.onload = initAll;
var xhr = false;
function initAll() {  same as previously except it calls getPix }
function getPix() {  xhr.open("GET", "flickrfeed.xml", true);
xhr.onreadystatechange = showPictures;  xhr.send(null);setTimeout("getPix()",5 * 1000); }
function showPictures() {
    var tempDiv = document.createElement("div");
    var tempDiv2 = document.createElement("div");
    if (xhr.readyState == 4) {
          if (xhr.status == 200) {
          tempDiv.innerHTML = xhr.responseText;
          var allLinks = tempDiv.getElementsByTagName("a");
          for (var i=1; i<allLinks.length; i+=2) {
                    tempDiv2.appendChild(allLinks[i].cloneNode(true)); }
          allLinks = tempDiv2.getElementsByTagName("a");
          var randomImg = Math.floor(Math.random() * allLinks.length);
    document.getElementById("pictureBar").innerHTML = allLinks[randomImg].innerHTML;
          } else { alert("There was a problem with the request " + xhr.status); } }  }
```

The call to getPix is placed in setTimeout which causes repeated execution, every 5 seconds;
An array of links of photographs is created, a random number computed, and use it as an index into the array

Browser Output


My Third Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
Done
0.296s

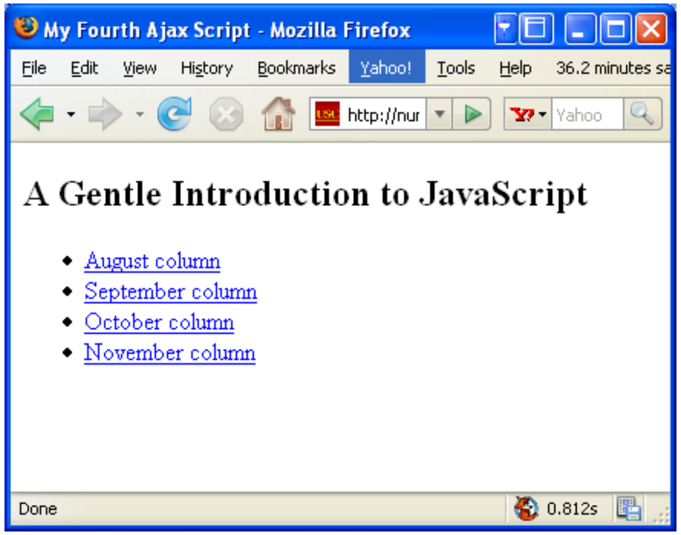

My Third Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
Done
0.296s


My Third Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
Done
0.296s

Three consecutive outputs
http://cs-server.usc.edu:45678/ajaxexamples/simple/script03.html

Fourth Ajax Example – Previewing Links

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Transitional//EN">
<html><head>
    <title>My Fourth Ajax Script</title>
    <link rel="stylesheet" rev="stylesheet"
    href="script04.css" />
    <script src="script04.js"
    type="text/javascript" language="Javascript">
    </script>
</head><body>
<h2>A Gentle Introduction to JavaScript</h2><ul>
    <li><a href="jsintro/2000-08.html">August
    column</a></li>
    <li><a href="jsintro/2000-09.html">September
    column</a></li>
    <li><a href="jsintro/2000-10.html">October
    column</a></li>
    <li><a href="jsintro/2000-11.html">November
    column</a></li>
</ul>
<div id="previewWin"> </div>
</body>
</html>
```


My Fourth Ajax Script - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help 36.2 minutes sa
http://nur
Yahoo
Done
0.812s

A Gentle Introduction to JavaScript

- August column
- September column
- October column
- November column

http://cs-server.usc.edu:45678/ajaxexamples/simple/script04.html

The stylesheet

```
#previewWin {
   background-color: #FF9;
   width: 400px;
   height: 100px;
   font: .8em arial, helvetica, sans-serif;
   padding: 5px;
   position: absolute;
   visibility: hidden;
   top: 10px;
   left: 10px;
   border: 1px #CC0 solid;
   clip: auto;
   overflow: hidden;
}

#previewWin h1, #previewWin h2 {
   font-size: 1.0em;
}
```

The javascript source

```
window.onload = initAll;
var xhr = false;
var xPos, yPos;
function initAll() {
    var allLinks = document.getElementsByTagName("a");
    for (var i=0; i< allLinks.length; i++) {
          allLinks[i].onmouseover = showPreview; } }
function showPreview(evt) { getPreview(evt); return false; }
function hidePreview() {
      document.getElementById("previewWin").style.visibility = "hidden"; }
function getPreview(evt) {
    if (evt) { var url = evt.target; }
    else { evt = window.event;  var url = evt.srcElement; }
    xPos = evt.clientX;  yPos = evt.clientY;
    if (window.XMLHttpRequest) {
          xhr = new XMLHttpRequest(); }
    else { if (window.ActiveXObject) {
          try {  xhr = new ActiveXObject("Microsoft.XMLHTTP"); }  catch (e) { }  } }
    if (xhr) { xhr.onreadystatechange = showContents;
          xhr.open("GET", url, true);  xhr.send(null);
    } else { alert("Sorry, but I couldn't create an XMLHttpRequest"); }  }
```

The javascript source cont'd

```
function showContents() {
    var prevWin = document.getElementById("previewWin");
    if (xhr.readyState == 4) {
          prevWin.innerHTML = (xhr.status == 200) ? xhr.responseText : "There was
    a problem with the request " + xhr.status;
          prevWin.style.top = parseInt(yPos)+2 + "px";
          prevWin.style.left = parseInt(xPos)+2 + "px";
          prevWin.style.visibility = "visible";
          prevWin.onmouseout = hidePreview; }}
```

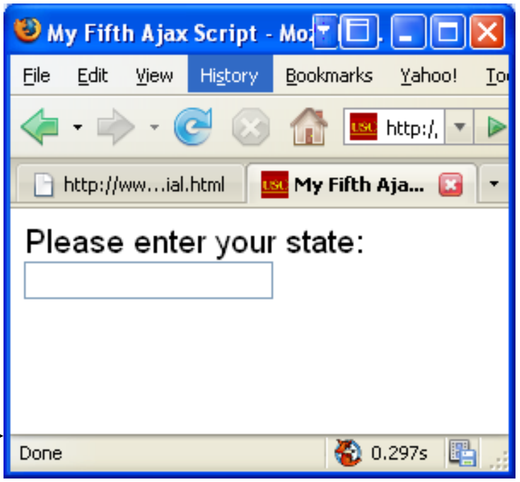
Notes: initall goes through all of the links and adds an onmouseover event;
showPreview( ) and hidePreview( ) are both needed; the latter sets the preview window back to hidden;
In getPreview( ), depending upon the browser, the URL is in either evt.target or in window.event.srcElement; the (x,y) position is extracted;
In showContents( ) the data is placed in prevWin.innerHTML from responseText;
The preview window is placed just below and to the right of the cursor position that triggered the call

Fifth Ajax Example, Auto Completion

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html><head><title>My Fifth Ajax Script</title>
   <link rel="stylesheet" rev="stylesheet" href="script05.css" />
   <script src="script05.js" type="text/javascript"
   language="Javascript">
   </script>
</head><body>
   <form action="#">
   Please enter your state:<br />
   <input type="text" id="searchField" autocomplete="off" /><br />
        <div id="popups"> </div>
   </form></body></html>
```

Autocomplete attribute is set to off to prevent browsers from trying to autocomplete the field

Initial screen


My Fifth Ajax Script - Mo
File Edit View History Bookmarks Yahoo! To
http:/
http://ww...ial.html
My Fifth Aja...
Done
0.297s
Please enter your state:

The stylesheet

```
body, #searchfield {
   font: 1.2em arial, helvetica, sans-serif;
}
.suggestions {
   background-color: #FFF;
   padding: 2px 6px;
   border: 1px solid #000;
}
.suggestions:hover {
   background-color: #69F;
}
#popups {
   position: absolute;
}
#searchField.error {
   background-color: #FFC;
}
```

The JavaScript Source

Onkeyup captures single keystrokes

The example uses the xml file listing the states

Here we read the list of states and place them in an array

```
window.onload = initAll;
var xhr = false;   var statesArray = new Array();
function initAll() {
    document.getElementById("searchField").onkeyup = searchSuggest;
    if (window.XMLHttpRequest) {  xhr = new XMLHttpRequest(); }
    else { if (window.ActiveXObject) {
          try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) { } }}
    if (xhr) {
          xhr.onreadystatechange = setStatesArray;
          xhr.open("GET", "us-states.xml", true); xhr.send(null);
    } else { alert("Sorry, but I couldn't create an XMLHttpRequest"); }}
function setStatesArray() {
    if (xhr.readyState == 4) {
          if (xhr.status == 200) {
                     if (xhr.responseXML) {
          var allStates = xhr.responseXML.getElementsByTagName("item");
                              for (var i=0; i<allStates.length; i++) {
                                        statesArray[i] =
    allStates[i].getElementsByTagName("label")[0].firstChild; } }  }
    else { alert("There was a problem with the request " + xhr.status); } }  }
```

The JavaScript Source cont'd

```
function searchSuggest() {
    var str = document.getElementById("searchField").value;
    document.getElementById("searchField").className = "";
    if (str != "") {
          document.getElementById("popups").innerHTML = "";
          for (var i=0; i<statesArray.length; i++) {
                    var thisState = statesArray[i].nodeValue;
                              if
    (thisState.toLowerCase().indexOf(str.toLowerCase()) == 0) {
          var tempDiv = document.createElement("div");
          tempDiv.innerHTML = thisState;
          tempDiv.onclick = makeChoice;
          tempDiv.className = "suggestions";
          document.getElementById("popups").appendChild(tempDiv); }   }
          var foundCt = document.getElementById("popups").childNodes.length;
          if (foundCt == 0) {
                    document.getElementById("searchField").className = "error"; }
          if (foundCt == 1) {
                    document.getElementById("searchField").value =
    document.getElementById("popups").firstChild.innerHTML;
                    document.getElementById("popups").innerHTML = ""; }   }   }
function makeChoice(evt) {
    var thisDiv = (evt) ? evt.target : window.event.srcElement;
    document.getElementById("searchField").value = thisDiv.innerHTML;
    document.getElementById("popups").innerHTML = ""; }
```

This routine is called on a key up; The value in the search field is first extracted; if nothing is entered, do nothing;

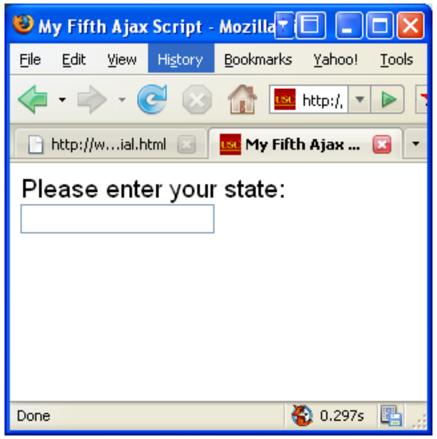
If indexof returns 0, then we have a hit;

Add a state to the list of possibilities

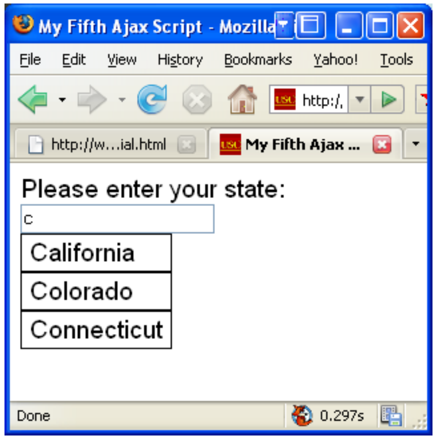
Foundct is the number of matches

Unique hit, place it in proper place

Browser Output


My Fifth Ajax Script - Mozilla
File Edit View History Bookmarks Yahoo! Tools
http://w...ial.html
My Fifth Ajax ...
Done
0.297s

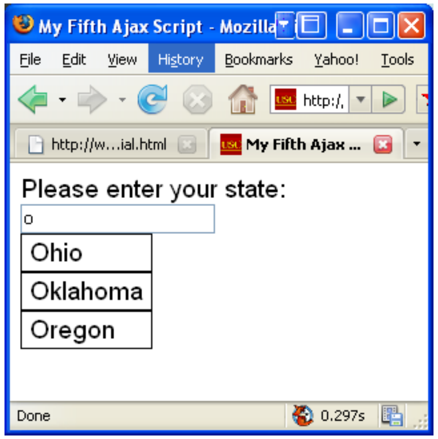
Please enter your state:


My Fifth Ajax Script - Mozilla
File Edit View History Bookmarks Yahoo! Tools
http://w...ial.html
My Fifth Ajax ...
Done
0.297s

Please enter your state:

c

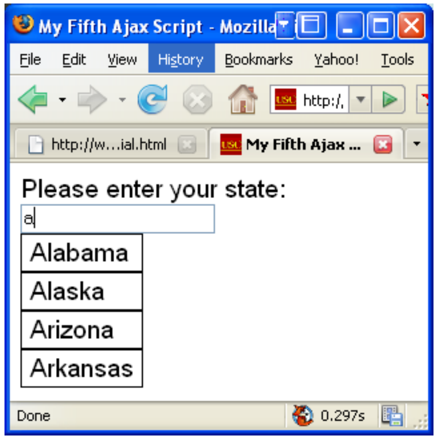
California
Colorado
Connecticut


My Fifth Ajax Script - Mozilla
File Edit View History Bookmarks Yahoo! Tools
http://w...ial.html
My Fifth Ajax ...
Done
0.297s

Please enter your state:

o

Ohio
Oklahoma
Oregon


My Fifth Ajax Script - Mozilla
File Edit View History Bookmarks Yahoo! Tools
http://w...ial.html
My Fifth Ajax ...
Done
0.297s

Please enter your state:

a

Alabama
Alaska
Arizona
Arkansas

Initial screen 3 examples

http://cs-server.usc.edu:45678/ajaxexamples/simple/script05.html

Ajax Slide Show Example

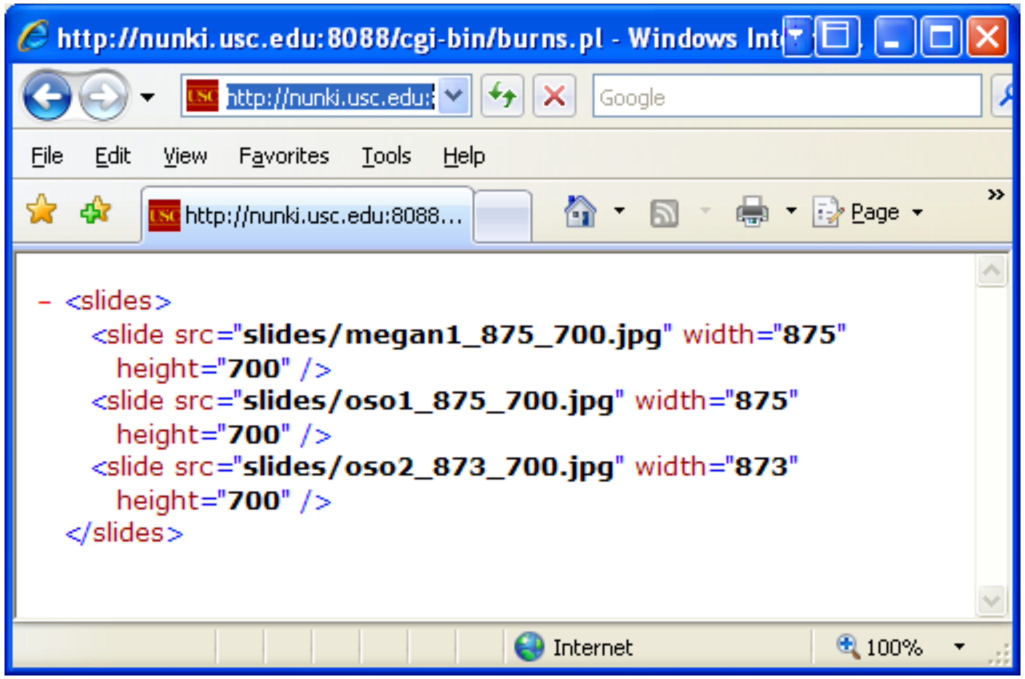
- We want to create an application that downloads a set of jpg images from a server and shows them in a slide show
- We want to avoid the delay that comes from downloading each slide one-at-a-time across the Internet
- We use Ajax to control the downloading of slides so that the slide show appears to work smoothly

PHP Script to Extract Slides

- we begin by creating a PhP script for accessing the jpg images
- this script can be thought of as providing a simple *web service*, namely delivering an xml file of images
- all images have file names of the form name_width_height.jpg where width and height are numerical values

```
<?php
header( "Content-type: text/xml" );
?>
<slides>
<?php
if ($handle = opendir('images')) {
while (false !== ($file = readdir($handle)))
{
        if ( preg_match( "/[.]jpg$/", $file ) ) {
                preg_match( "/_(\d+)_(\d+)[.]/", $file, $found );
?>
<slide src="images/<?php echo $file; ?>"
  width="<?php echo $found[1]; ?>"
  height="<?php echo $found[2]; ?>" /><?php echo( "\n" ); ?>
<?php
        }
}
closedir($handle);
}
?>
</slides>
```

Browser output


http://nunki.usc.edu:8088/cgi-bin/burns.pl - Windows Int
http://nunki.usc.edu:
Google
File Edit View Favorites Tools Help
http://nunki.usc.edu:8088...
Page
Internet
100%

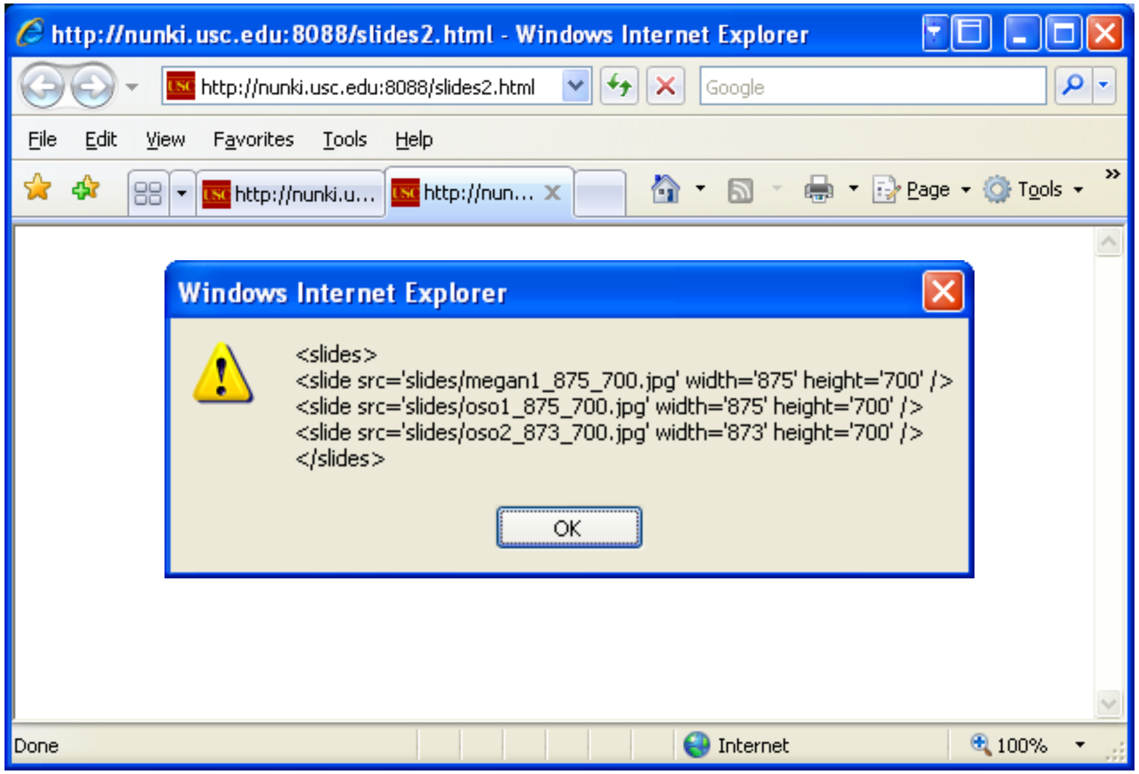
```
- <slides>
    <slide src="slides/megan1_875_700.jpg" width="875"
      height="700" />
    <slide src="slides/oso1_875_700.jpg" width="875"
      height="700" />
    <slide src="slides/oso2_873_700.jpg" width="873"
      height="700" />
  </slides>
```

Simple Program to Read Data from the service using Ajax Connection

```
<html> <body> <script>
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 && req.responseXML != null)
   { alert( req.responseText ); } }
function loadXMLDoc( url ) {
  req = false;
  if(window.XMLHttpRequest) { try { req = new XMLHttpRequest();
    } catch(e) { req = false; }
} else if(window.ActiveXObject)
  { try { req = new ActiveXObject("Msxml2.XMLHTTP");
    } catch(e) {
   try { req = new ActiveXObject("Microsoft.XMLHTTP");
    }  catch(e) { req = false; } } }
if(req) {
  req.onreadystatechange = processReqChange;
  req.open("GET", url, true);
  req.send(""); } }
loadXMLDoc( "http://nunki.usc.edu:8088/cgi-bin/burns.pl" );
</script> </body> </html>
```

loadXMLDoc determines how to invoke XMLHttpRequest and then sends an Ajax GET request To execute burns.pl; That request goes off asynchronously to retrieve the page and return the result. When the request is complete, the processReqChange function is called with the Result which displays the value of the responseText function in an alert window.

Browser Output


http://nunki.usc.edu:8088/slides2.html - Windows Internet Explorer
http://nunki.usc.edu:8088/slides2.html
Google
File Edit View Favorites Tools Help
http://nunki.u...
http://nun...
Page
Tools
Windows Internet Explorer
OK
Done
Internet
100%

```
<slides>
<slide src='slides/megan1_875_700.jpg' width='875' height='700' />
<slide src='slides/oso1_875_700.jpg' width='875' height='700' />
<slide src='slides/oso2_873_700.jpg' width='873' height='700' />
</slides>
```

Notes on the Previous Program

- the XML data is successfully coming back from the server.
- the URL is an absolute path, domain name and all. That's the only valid URL style for Ajax. The server code that writes the Ajax JavaScript code always creates valid, fully formed URLs.
- Ajax security precautions - The JavaScript code can't ask for just any URL. The URL must have the same domain name as the page.
  - you can't render HTML from www.mycompany.com, and then have the script retrieve data from data.mycompany.com. Both domains must match exactly, including the sub-domains.
- the code in loadXMLDoc, Pre-version 7 Internet Explorer doesn't have the XMLHTTPRequest object type built in. So, one must use Microsoft ActiveX controls.
- Finally, in the processReqChange function,
  - The readyState value of **4** means that the transaction is complete.
  - The status value of **200** means that the page is valid.

Creating HTML dynamically

- Now we extend the current example by having the processReqChange function create an HTML table with the results of the XML request from the server.
- In that way, one can test that it is possible to read the XML and create HTML from it dynamically.

Enhanced Test Page

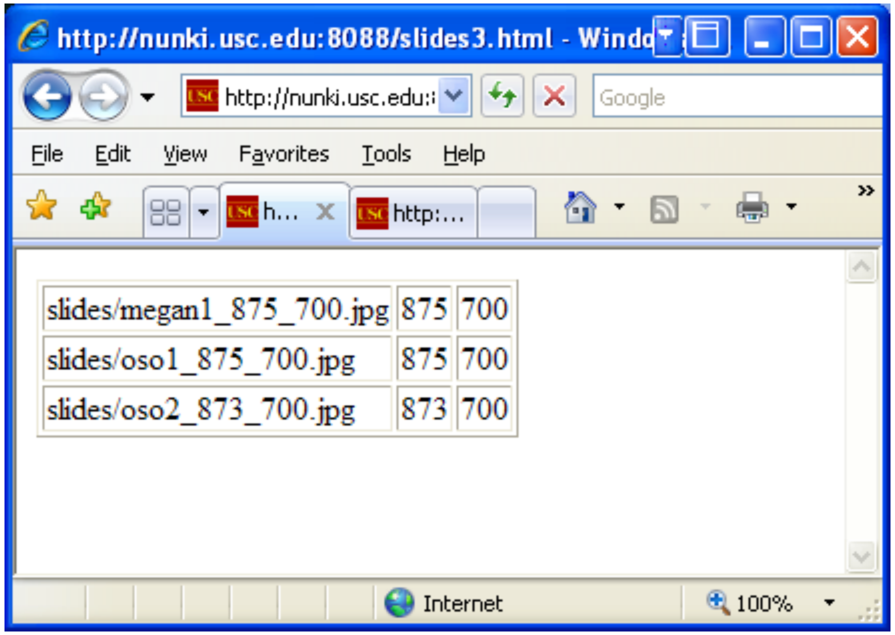
```
<html> <body> <table border=1> <tbody id="dataTable"> </tbody> </table>
<script> function processReqChange() {
  if (req.readyState == 4 && req.status == 200 && req.responseXML != null) {
      var dto = document.getElementById( 'dataTable' );
      var items = [ ];
      var nl = req.responseXML.getElementsByTagName( 'slide' );
      for( var i = 0; i < nl.length; i++ ) {
         var nli = nl.item( i );
         var src = nli.getAttribute( 'src' ).toString();
var width = parseInt( nli.getAttribute( 'width' ).toString() );
var height = parseInt( nli.getAttribute( 'height' ).toString() );
         var trNode = document.createElement( 'tr' );
         var srcNode = document.createElement( 'td' );
         srcNode.innerHTML = src;
         trNode.appendChild( srcNode );
         var widthNode = document.createElement( 'td' );
         widthNode.innerHTML = width.toString();
         trNode.appendChild( widthNode );
```

The updated processReqChange code now looks at the responseXML object instead of the responseText text. In addition, it uses getElementsByTagName to access all the <slide> tags. From there, it parses the src, width, and height attributes and uses the createElement method on the document object to create rows and cells to hold the data. This method of using the createElement method is far more robust than the method in which you create an HTML string with the contents of the table and use innerHTML to add the data to an existing element.

Enhanced Test Page (cont'd)

```
  var heightNode = document.createElement( 'td' );
   heightNode.innerHTML = height.toString();
   trNode.appendChild( heightNode );
   dto.appendChild( trNode ); }
   load_slides( items );
   start_slides(); } }
function loadXMLDoc( url ) {
  req = false;
  if(window.XMLHttpRequest) {
      try { req = new XMLHttpRequest();
          } catch(e) {
                  req = false; }
        } else if(window.ActiveXObject) {
                try { req = new ActiveXObject("Msxml2.XMLHTTP");
         } catch(e) {
               try { req = new ActiveXObject("Microsoft.XMLHTTP");
        } catch(e) { req = false; } } }
if(req) { req.onreadystatechange = processReqChange;
req.open("GET", url, true);
req.send(""); } }
loadXMLDoc( "http://nunki.usc.edu:8088/cgi-bin/burns.pl" );
</script> </body> </html>
```

Browser Output


http://nunki.usc.edu:8088/slides3.html - Windo
http://nunki.usc.edu:
Google
File Edit View Favorites Tools Help
h...
http:...
Internet
100%

slides/megan1_875_700.jpg	875	700
slides/oso1_875_700.jpg	875	700
slides/oso2_873_700.jpg	873	700

Some References

- Ajax: A New Approach to Web Applications http://adaptivepath.org/ideas/ajax-new-approach-web-applications/
- Ajax (programming) – Wikipedia: http://en.wikipedia.org/wiki/AJAX
- Using the XML HTTP Request object: http://jibbering.com/2002/4/httprequest.html
- XMLHttpRequest & Ajax Working Examples: http://www.fiftyfoureleven.com/resources/programming/xmlhttprequest/examples
- Very Dynamic Web Interfaces: http://www.xml.com/pub/a/2005/02/09/xml-http-request.html

Ajax Enabled Technologies (Toolkits)

- Ruby on Rails:
  http://www.rubyonrails.org/
- Microsoft ASP.NET Atlas:
  http://www.asp.net/ajax/
- Eclipse / IBM -- The AJAX Toolkit Framework (ATF):
  http://www.eclipse.org/atf/
- Kabuki Ajax Toolkit:
  http://www.zimbra.com/community/kabuki_ajax_toolkit_download.html
- Yahoo! User Interface Library (**DEPRECATED**):
  http://developer.yahoo.net/yui/

Browser Security Features

Credits

- The following material is based on the google wiki, Browser Security Handbook:

  https://code.google.com/p/browsersec/wiki/Part1

  https://code.google.com/p/browsersec/wiki/Part2

Part1 Outline

- Basic concepts behind web browsers
  - Uniform Resource Locators
    - Unicode in URLs
  - True URL schemes
  - Pseudo URL schemes
  - Hypertext Transfer Protocol
  - Hypertext Markup Language
    - HTML entity encoding
  - Document Object Model
  - Browser-side Javascript
    - Javascript character encoding
  - Other document scripting languages
  - Cascading stylesheets
    - CSS character encoding
  - Other built-in document formats
  - Plugin-supported content

Part2 Outline

- Standard browser security features
  - Same-origin policy
    - Same-origin policy for DOM access
    - Same-origin policy for XMLHttpRequest
    - Same-origin policy for cookies
    - Same-origin policy for Flash
    - Same-origin policy for Java
    - Same-origin policy for Silverlight
    - Same-origin policy for Gears
    - Origin inheritance rules
    - Cross-site scripting and same-origin policies
  - Life outside same-origin rules
    - Navigation and content inclusion across domains
    - Arbitrary page mashups (UI redressing)
    - Gaps in DOM access control
    - Privacy-related side channels
  - Various network-related restrictions
    - Local network / remote network divide
    - Port access restrictions
    - URL scheme access rules
    - Etc

Same-origin policy for DOM access

- the term "same-origin policy" most commonly refers to a mechanism that governs the ability for Javascript and other scripting languages to access DOM properties and methods across domains
- the same-origin model attempts to ensure proper separation between unrelated pages, and serve as a method for sandboxing potentially untrusted or risky content within a particular domain

Three-Step Decision Process

- the model boils down to this three-step decision process:

1. If protocol, host name, and - for browsers other than Microsoft Internet Explorer - port number for two interacting pages match, access is granted with no further checks.
2. Any page may set the ***document.domain*** parameter to a right-hand, fully-qualified fragment of its current host name (e.g., ***foo.bar.example.com*** may set it to ***example.com***, but not ***ample.com***). If two pages explicitly and *mutually* set their respective ***document.domain*** parameters to the same value, and the remaining same-origin checks are satisfied, access is granted.
3. If neither of the above conditions is satisfied, access is denied.

Drawbacks of Same-Origin Policy

- once any two legitimate subdomains in ***example.com***, e.g. ***www.example.com*** and ***payments.example.com***, choose to cooperate, any other resource in that domain, such as ***user-pages.example.com***, may then set its own ***document.domain*** likewise, and arbitrarily mess with ***payments.example.com***. This means that in many scenarios, ***document.domain*** may not be used safely at all.
- Whenever ***document.domain*** cannot be used - either because pages live in completely different domains, or because of the above problem - legitimate client-side communication between, for example, embeddable page gadgets, is completely forbidden in theory, and in practice very difficult to arrange
- Whenever tight integration of services within a single host name is pursued to overcome these communication problems, because of the inflexibility of same-origin checks, there is no usable method to sandbox any untrusted or particularly vulnerable content to minimize the impact of security problems.

Special Cases that Are *Omitted* From the Policy

- The ***document.domain*** behavior when hosts are addressed by IP addresses, as opposed to fully-qualified domain names, is not specified.
- The ***document.domain*** behavior with extremely vague specifications (e.g., ***co.uk***) is not specified.
- The algorithms of context inheritance for pseudo-protocol windows, such as ***about:blank***, are not specified.
- The behavior for URLs that do not meaningfully have a host name associated with them (e.g., ***file://***) is not defined, causing **some browsers** to permit locally saved files to access every document on the disk or on the web; users are generally not aware of this risk, potentially exposing themselves.
- The behavior when a single name resolves to vastly different IP addresses (for example, one on an internal network, and another on the Internet) is not specified, permitting various attacks and tricks

Same-origin policy for XMLHttpRequest

- security-relevant features provided by ***XMLHttpRequest***
  - The ability to specify an arbitrary HTTP request method (via the ***open()*** method),
  - The ability to set custom HTTP headers on a request (via ***setRequestHeader()***),
  - The ability to read back full response headers (via ***getResponseHeader()*** and ***getAllResponseHeaders()***),
  - The ability to read back full response body as Javascript string (via ***responseText*** property).

Checks on XMLHttpRequest

- The set of checks implemented in all browsers for ***XMLHttpRequest*** is a close variation of DOM same-origin policy, with the following changes:
- Checks for ***XMLHttpRequest*** targets do not take ***document.domain*** into account, making it impossible for third-party sites to mutually agree to permit cross-domain requests between them.
- In some implementations, there are additional restrictions on protocols, header fields, and HTTP methods for which the functionality is available, or HTTP response codes which would be shown to scripts (see later).

Cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) allows many resources (e.g, fonts, JavaScript, etc.) on a web page to be requested across domains.

In particular, AJAX calls can use XMLHttpRequest across domains. Such “cross-domain” requests would otherwise be forbidden by web browsers.

The CORS standard adds new HTTP headers. To initiate a CORS request, a browser sends the request with an “Origin” HTTP header. Suppose a page from http://www.social-network.com attempts to access user data from online-personal-calendar.com. If the browser supports CORS, this header is sent:

Origin: http://www.social-network.com

If the server at online-personal-calendar.com allows the request, it sends an Access-Control-Allow-Origin (ACAO) header in the response. The value of the header indicates what origin sites are allowed. For example:

Access-Control-Allow-Origin: http://www.social-network.com

If the server does not allow the CORS request, the browser will deliver an error instead of the online-personal-calendar.com response. Firefox 3.5+, Safari 4+, Chrome3+, IE 10+, Opera 12+ support CORS. See:

    https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
    http://enable-cors.org/server_apache.html

Problems and Drawbacks with Cookies

- **Privacy issues:** the chief concern with the mechanism was that it permitted scores of users to be tracked extensively across any number of collaborating domains without permission
- **Problems with ccTLDs:** the specification did not account for the fact that many country-code TLDs are governed by odd or sometimes conflicting rules. For example, ***waw.pl***, ***com.pl***, and ***co.uk*** should be all seen as generic, functional top-level domains,
- **Problems with conflict resolution:** when two identically named cookies with different scopes are to be sent in a single request, there is no information available to the server to resolve the conflict and decide which cookie came from where, or how old it is
- **Problems with certain characters:** just like HTTP, cookies have no specific provisions for character escaping, and no specified behavior for handling of high-bit and control characters
- **Problems with cookie jar size:** standards do relatively little to specify cookie count limits or pruning strategies. Various browsers may implement various total and per-domain caps, and the behavior may result in malicious content purposefully disrupting session management, or legitimate content doing so by accident.

Same-Origin Policy for Flash

- Adobe Flash is a plugin believed to be installed on about 99% of all desktops, incorporates a security model generally inspired by browser same-origin checks.
- Flash applets have their security context derived from the URL they are loaded from
- Within this realm, permission control follows the same basic principle as applied by browsers to DOM access:
  - protocol, host name, and port of the requested resource is compared with that of the requestor, with universal access privileges granted to content stored on local disk.

Same-Origin Policy for Java

- Much like Adobe Flash, Java applets, reportedly supported on about 80% of all desktop systems, follow the basic concept of same-origin checks applied to a runtime context derived from the site the applet is downloaded from
- the following permissions are available to Java applets:
  - The ability to interact with Javascript on the embedding page through the JSObject API, with no specific same-origin checks. This mechanism is disabled by default, but may be enabled with the ***MAYSCRIPT*** parameter within the ***<APPLET>*** tag.
  - In some browsers, the ability to interact with the embedding page through the DOMService API. The documentation does not state what, if any, same-origin checks should apply; based on the aforementioned tests, no checks are carried out, and cross-domain embedding pages may be accessed freely with no need for ***MAYSCRIPT*** opt-in. This directly contradicts the logic of ***JSObject*** API.
  - The ability to send same-origin HTTP requests using the browser stack via the URLConnection API, with virtually no security controls, including the ability to set ***Host*** headers, or insert conflicting caching directives. On the upside, it appears that there is no ability to read 30x redirect bodies or ***httponly*** cookies from within applets.
  - The ability to initiate unconstrained TCP connections back to the originating host, and that host only, using the Socket API. These connections do not go through the browser, and are not subject to any additional security checks (e.g., ports