Coding AJAX Applications
(Using XHTML, JavaScript and COBOL on z/OS)

- **Introduction to AJAX**
- **Experiment 1 - Invoking a special CGI**
- **Experiment 2 - Referencing an existing HTML fragment**
- **Experiment 3 - Referencing an existing XML document**
- **Summary and conclusions**

by Steve Comstock

The Trainer's Friend, Inc.
http://www.trainersfriend.com
303-393-8716
steve@trainersfriend.com

v2.2

The following terms that may appear in these course materials are trademarks or registered trademarks:

Trademarks of the International Business Machines Corporation:

AD/Cycle, AIX, AIX/ESA, Application System/400, AS/400, BookManager, CICS, CICS/ESA, COBOL/370, COBOL for MVS and VM, COBOL for OS/390 & VM, Common User Access, CORBA, CUA, DATABASE 2, DB2, DB2 Universal Database, DFSMS, DFSMSds, DFSORT, DOS/VSE, Enterprise System/3090, ES/3090, 3090, ESA/370, ESA/390, Hiperbatch, Hiperspace, IBM, IBMLink, IMS, IMS/ESA, Language Environment, MQSeries, MVS, MVS/ESA, MVS/XA, MVS/DFP, NetView, NetView/PC, Object Management Group, Operating System/400, OS/400, PR/SM, OpenEdition MVS, Operating System/2, OS/2, OS/390, OS/390 UNIX, OS/400, QMF, RACF, RS/6000, SOMobjects, SOMobjects Application Class Library, System/370, System/390, Systems Application Architecture, SAA, System Object Model, TSO, VisualAge, VisualLift, VTAM, VM/XA, VM/XA SP, WebSphere, z/OS, z/VM, z/Architecture, zSeries

Trademarks of Microsoft Corp.: Microsoft, Windows, Windows NT, Visual Basic, Microsoft Access, MS-DOS, Windows XP

Trademarks of Micro Focus Corp.: Micro Focus

Trademarks of America Online, Inc.: America Online

Trademarks of Quercus Systems: Personal REXX, REXXTERM

Trademark of Chicago-Soft, Ltd: MVS/QuickRef

Trademark of Crystal Computer Services: Crystal Reports

Trademark of Phoenix Software International: (E)JES

Registered Trademarks of Institute of Electrical and Electronic Engineers: IEEE, POSIX

Registered Trademarks of Corel Corporation: Corel, CorelDRAW, Corel VENTURA

Registered Trademark of Oracle Corporation: Oracle

Registered Trademark of The Open Group: UNIX

Trademark of Sun Microsystems, Inc.: Java

Registered Trademark of Linus Torvalds: LINUX

Registered Trademark of Unicode, Inc.: Unicode

Trademarks held on behalf of World Wide Web Consortium: W3C, XHTML, XSL, WebFonts

Section Preview

- Introduction to AJAX
  - AJAX - Asynchronous JavaScript with XML
  - The XMLHttpRequest Object
  - Coding For AJAX
  - AJAX Requests and Responses
  - Using the Response From an AJAX Request
  - Coding for Older Browsers
  - Making Requests
  - Using Style Sheets
  - Control Flow

AJAX - Asynchronous JavaScript with XML

AJAX is a term coined by Jesse James Garrett of Adaptive Path in 2005. The term has become popular even though it is totally misleading:

* Asynchronous - meaning work is being started and the requestor then does not wait for it to complete before doing other tasks while the requested work runs on its own;

    Actually, AJAX code may be synchronous or asynchronous

* JavaScript - meaning the requesting is being done from a program written in JavaScript; however, other client side scripting languages may be used

* XML - meaning the data exchanged is in XML format; the truth is that data being exchanged can be just text, or HTML or XML or XHTML - but it must be text.

The goal of AJAX coding is simply to allow interactive Web work without having to reload a page. That is, whenever possible just update the current page instead of having to retransmit or re-emit a new version of a web page.

Note that AJAX is not a product, but rather an approach that uses some commonly available tools and techniques.

This paper is just the result of my explorations and experiments; it's pretty thorough but not exhaustive.

Special thanks to **John McKown** of HealthMarkets who, in October 2006 published a note on IBM-Main indicating he got AJAX working on z/OS and on request he forwarded me his files, which I used for as a starting point for learning. (This led me to the W3C and Wikipedia articles on XMLHttpRequest, which filled in a lot of the details.)

The XMLHttpRequest Object

- **The service that allows AJAX to work is the XMLHttpRequest object**
  - **This is a request for HTTP work from an HTML page**
- **This is not currently an official standard**
  - **Originally provided by MicroSoft in Internet Explorer, adapted by other browsers**
  - **A work in progress at W3C (World Wide Web Consortium); see http://www.w3.org/TR/XMLHttpRequest/**
  - **Already enhancements are being proposed; see http://dev.w3.org/2006/webapi/XMLHttpRequest-2/**
  - **Also lookng to be a part of the HTML 5 standard when it is approved; see http://dev.w3.org/html5/spec/Overview.html**

The XMLHttpRequest Object, 2

❒ **The XMLHttpRequest object has the following attributes:**

- **readyState - short integer identifying the current state of the object; in code may test for the integer or the case-sensitive text below:**
  - ✗ UNSENT = 0; initial state; object created
  - ✗ OPENED = 1; request has been made
  - ✗ HEADERS_RECEIVED = 2; response headers have been received
  - ✗ LOADING = 3; response data is being returned
  - ✗ DONE = 4; request has completed
- **onreadystatechange - event triggered each time the request progresses from one ready state to another; must have an event handler for this**
- **responseText - the response from the server as text string**
- **responseXML - the response from the server as an XML document**
- **status - short integer containing the HTTP status; usually 200 for successful, anything else is an error of some kind**
- **statusText - a string containing the text related to the status number, for example "Document follows"**

The XMLHttpRequest Object, 3

❒ **The XMLHttpRequest object has the following methods available:**

- ♦ **open - specify HTTP request, one of GET, POST, HEAD, PUT, DELETE, OPTIONS - initialize the object**

  ✗ I look exclusively at GET here

- ♦ **setRequestHeader - identify MIME type of data in a send request (if any data transmitted)**
- ♦ **send - initiate the request; possibly passing additional data**
- ♦ **abort - terminate the request if the user cancels**
- ♦ **getResponseHeader - retrieve the value of a specified header**
- ♦ **getAllResponseHeaders - returns all the HTTP headers as a single string, separated by UTF-16 CRLF**

❒ **The XMLHttpRequest object may raise the following possible exceptions:**

- ♦ **SYNTAX_ERR (ExceptionCode 12)**
- ♦ **INVALID_STATE_ERR (ExceptionCode 11)**
- ♦ **SECURITY_ERR (ExceptionCode 18)**
- ♦ **NOT_SUPPORTED_ERR (ExceptionCode 9)**
- ♦ **ABORT_ERR (ExceptionCode 20)**
- ♦ **NETWORK_ERR (ExceptionCode 19)**

Coding For AJAX

❒ **The key components needed in an AJAX application**

- ♦ **HTML or XHTML page with these scripting elements:**

  ✗ **Variable definitions**, at least one for an XMLHttpRequest object:

```
var sendReq = new XMLHttpRequest;
```

  ✗ A short **script to build and send the request**; e.g.:

```
function sendReqst()
   {
    sendReq.onreadystatechange=getResponse;
    sendReq.open("GET","cgioutp.xml");
    sendReq.send(null);
   }
```

  ✗ A **script to handle the response**; e.g.:

```
function getResponse()
   {
    if (sendReq.readyState == 4)
       {
        if (sendReq.status == 200)
          {
/* ... handle request here ... */
          }
   }
```

  ✗ A **control that allows the user to trigger the request**, maybe:

```
<input type="button" value="GO" onclick="sendReqst();"/>
```

AJAX Requests and Responses

- **Although you can make almost any HTTP request, the GET request seems most useful**

- **The item you are requesting may be either**

  - **A file (must be text of some kind, not an image file, say, or an audio file)**

    - ✗ If you want to get an image file sent, for example, you can request a text file that contains a valid img element

  - **A CGI program that returns plain text or an HTML or XML fragment or an entire document**

- **The reply is accessed through the responseText or responseXML attributes**

  - **If response is a valid XML or XHTML file, it will be found in both attributes (the text of the file in responseText, or as an Object of type XMLDocument in responseXML)**

  - **If response is a text file that is not valid XML, it will be in responseText but responseXML will be null**

Using the Response From an AJAX Request

❒ **I built three experiments and they are described briefly here:**

- **GET a CGI - I coded a CGI to accept a part number and use this to look up a value in a VSAM KSDS, returning just three values, which appear in the single responseText attribute**

  ✗ Then I displayed these values in the original form and allowed the user to calculate a price for any number of these items

- **GET a text file that contained just an img element**

  ✗ Then I inserted this element into a paragraph tag and let the browser go get the image

- **GET an XHTML file I had hanging around**

  ✗ Then extract an element from the retrieved document, and used it to replace an existing element in the original document

❒ **We'll examine each of these in turn, but first, a few common notes...**

Coding for Older Browsers

- All browsers I used to test these examples handled the XMLHttpRequest coding fine
  - But older browsers may not implement this
- Many examples in the literature show falling back to an older version, for MicroSoft Internet Explorer anyway, structured this way:

  Set a variable to represent the request object:

```
var sendReq = getXMLHttpRequest();
```

  Use a function to populate the variable, with try and catch logic:

```
function getXMLHttpRequest()
   {
      try {return new XMLHttpRequest(); } catch(e) {};
      try {return new ActiveXObject("Msxml2.XMLHTTP"); }
                                          catch(e) {};
      try {return new ActiveXObject("Microsoft.XMLHTTP"); }
                                          catch(e) {};
      return null;
   }
```

- I found this convolution to be unnecessary - but all my testing browsers were fairly recent versions
  - If you need to write code for older browsers, decide if you will use the above kind of logic, or possibly invoke an older style of interaction that bypasses the AJAX approach (since without this functionality the AJAX approach won't work anyway) if the user's browser does not support XMLHttpRequest

Making Requests

- **So the general approach that worked for all my examples was the one described earlier (see page 8)**

- **The open() method has two required operands, and three optional ones**

  - **HTTP request - in my experiments always "GET"**

  - **URI - identify the file you wish to get; in my experiments, one of:**

    - "/SCOMSTO/ajvcob?order_string="+order_string (the name of a CGI followed by a standard CGI parameter string)

    - "pic1.html" (the name of a file to transfer from the current working directory)

    - "cgioutp.xml" (again, the name of a file, but this time an XML document)

  - **{true|false} - should the request be asynchronous (true - the default) or synchronous (false); tradeoffs discussed shortly**

  - *userid* **- specify the id of a user this request should run under**

  - *password* **- the password for the above user**

    - I took the default for ***userid*** and ***password*** and it always worked for me

Using Style Sheets

- **The use of style sheets to enable the separation of content from representation is an ongoing direction of W3C**
  - **Specifically, the use of CSS (Cascading Style Sheets) to specify formatting as much as possible**
- **For all my experiments I used a single sytle sheet, built just for this work**
  - **Named** ajax.css **and stored in my html page directory; its contents:**

```
h1 {color: red;  font-size: 16pt; text-align: center; }

h2 {color: blue; font-size: 12pt; margin-left: .2in;
                                  text-align: left }

caption {color: red; font-size: 14pt; text-align: center }

div.getdata {border: groove blue 5px; padding: 2px;
             width: 95% }

div.getorder {border: groove green 5px; padding: 3px;
              width: 55%; }

div.feedback {border: groove red 5px; padding: 2px;
              width: 65%; }

div.instruction {border: groove red 3px; padding: 2px; }

div.tablebox {border: solid 2px; }

p {font-family: monospace; font-size: 10pt;
               font-weight:bold}

p.red {font-family: monospace; font-size: 10pt;
       font-weight:bold; color:red }

input {font-family: monospace; font-size: 10pt}

hr {color: #00FF00; }
```

Control Flow

❒ **When an XMLHttpRequest is set up (**open() **), the object is populated**

❒ **When the request is made (**send()**)**

- **If the third operand is** true **(request should be asynchronous), the onreadystatechange event listener will be invoked each time the state changes**

  ✗ The event handler for this will presumably check for a ready state of 4 and a successful completion

  ➢ Handle the result if successful

  ➢ lissue error messages if not successful

```
function sendReqst()
   {
    sendReq.onreadystatechange=getResponse;
    sendReq.open("GET","cgioutp.xml",true);
    sendReq.send(null);
   }
```

- **If the third operand is** false **(request should be synchronous), the event listener is not invoked - therefore you need to add code to invoke the routine for end-of-request handling explicitly**

```
function sendReqst()
   {
    sendReq.open("GET","cgioutp.xml",false);
    sendReq.send(null);
    getResponse();
   }
```

Control Flow, 2

❒ **I found the first case preferable and so used 'true' all the time**

- **In our first experiment, I put code in the event handler to display a progress bar (just used an** hr **element and increased the width based on the readyState value)**

  ✗ This is not very granular, but it was just to demonstrate one way to do it

❒ **Note that for all our experiments I used embedded scripts**

- **In some situations it's probably better to have external scripts (so you can share and maintain code more easily), but I thought it best to reduce the number of files to describe**

❒ **And now to explore the various experiments ...**

❒ **Examples in this paper were tested with**

- **Mozilla Firefox (5.0) - our benchmark browser, used in the sample displays in this paper unless otherwise stated**
- **Opera (10.53)**
- **Safari(4.0.5)**
- **Google Chrome (12.0.742.100)**
- **Internet Explorer (8.0.6001.18702).**

Section Preview

- **Experiment 1 - invoking a special CGI**
  - **The Setup**
  - **The Initial Page**
  - **In Progress**
  - **Returned Results**
  - **Calculating a Price**
  - **The Base Page: ajaxvcob.html**
  - **The COBOL CGI, AJVCOB**
  - **The encodeURIComponent Function**

Experiment 1 - The Setup

❒ **This is our most complex example**

- **The premise is to display a page where the user can request information (description, unit price, and quantity available) about an item in our inventory, based on the item's part number**

- **The request triggers an XMLHttpRequest to a CGI, passing the requested part number**

  ✗ This CGI, ajvcob, is a COBOL program that:

  - ➢ Accepts the passed part number
  - ➢ Allocates a VSAM file
  - ➢ Opens the VSAM file
  - ➢ Looks up the part number in the VSAM file and returns one of
    - ➤ An error mesage that begins with '<h2>'
    - ➤ A response string formatted as:

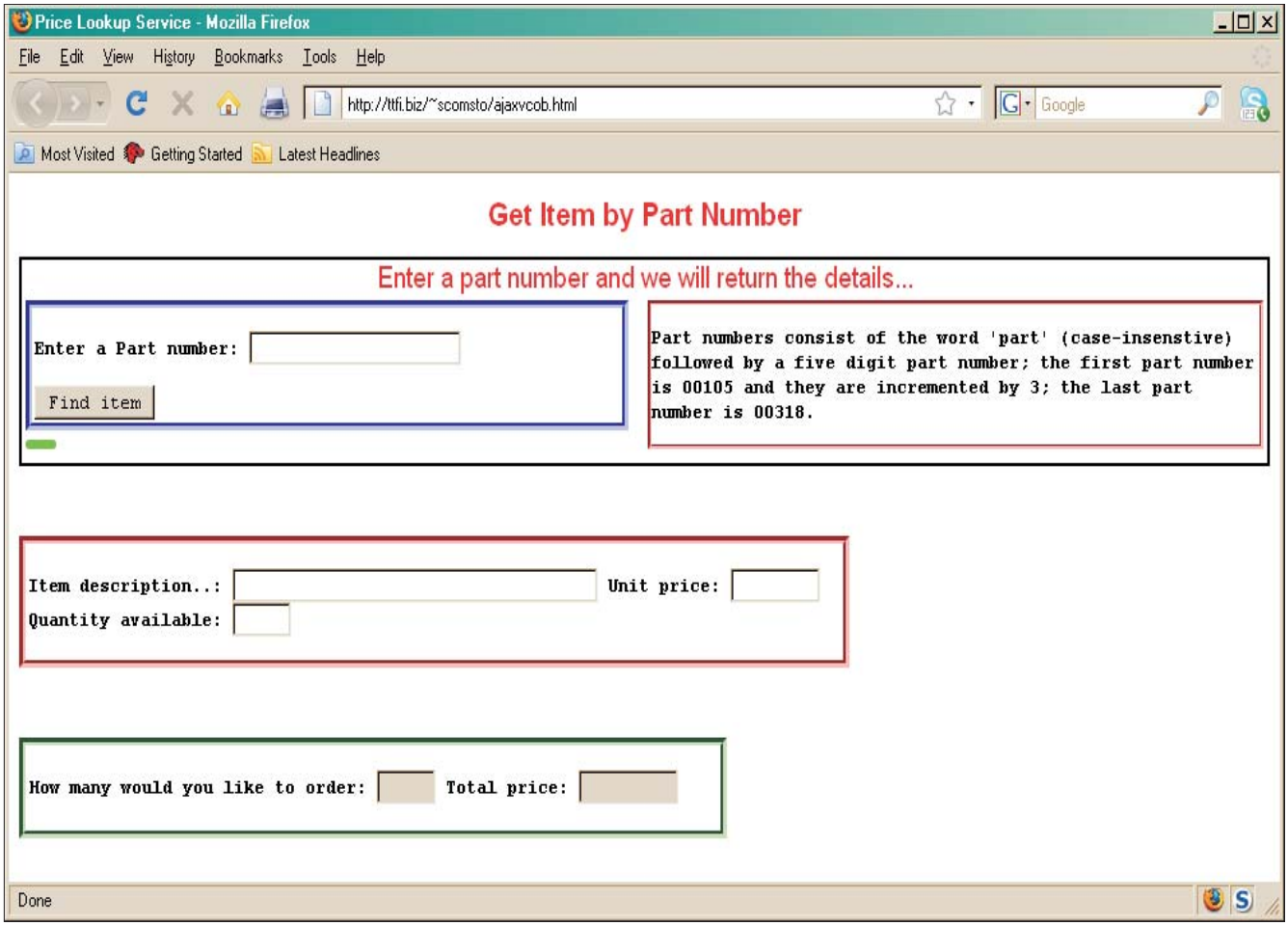
      descr=*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx* **unpr=***9999.999* **qty=***99999*
  - ➢ Closes the VSAM file
  - ➢ Frees the VSAM file
  - ➢ Returns

- **Back in the original page, I accept the response in responseText and parse it into description, unit price, and quantity input controls, and activate the qty and price fields**

- **When the user enters a number into the qty input control, a script on the page calculates the total price**

The Initial Page

❒ **The first page, ajaxvcob.html, renders this way:**

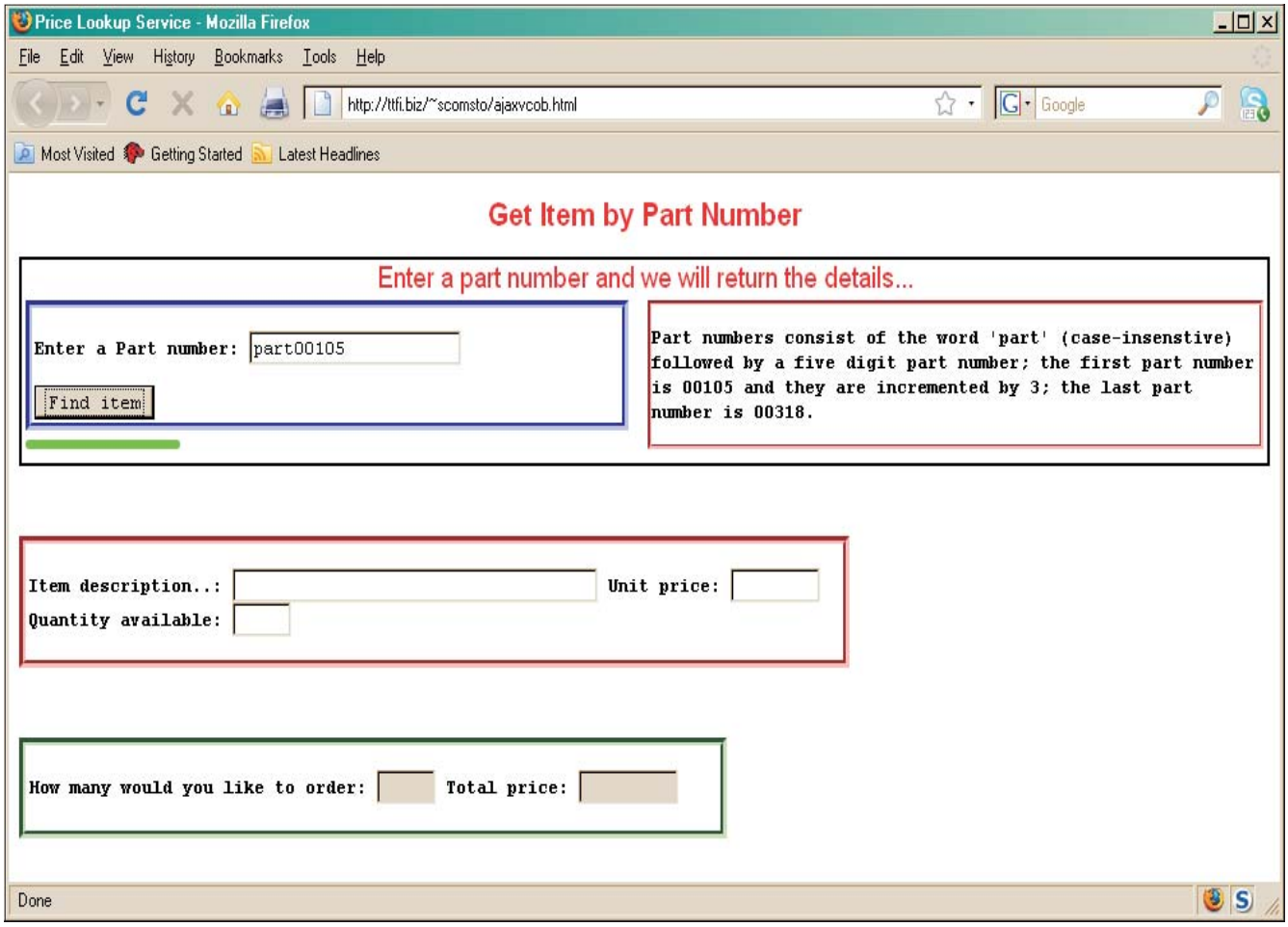


❒ **Notice the three distinct areas, top to bottom:**

- **Area to specify the requested part number (notice the horizontal line below the "Find item" button: this is the display of the** hr **tag I used to generate a progress bar)**
- **Area to view the result**
- **Area to request a quantity and see the price (currently inactive)**

In Progress

❒ **Here I have entered a part number and clicked "Find item" and the progress bar has expanded a bit:**

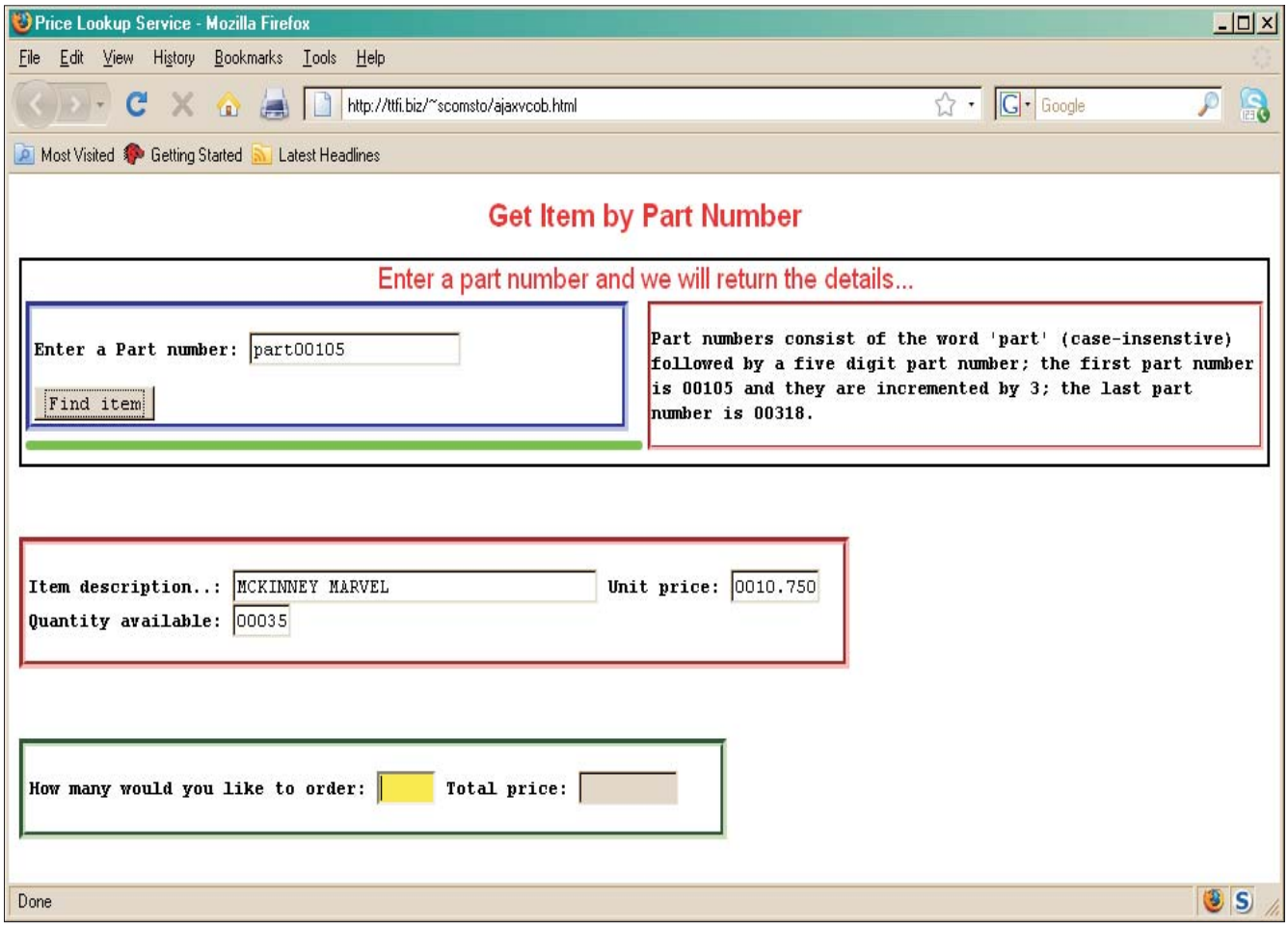


- **Since this experiment invokes a COBOL CGI that does a lot of work (Allocate, Open, Read, display, Close, Free), the process takes a little bit of time**

  ✗ More real world applications might work with files in the HFS or a database or some other approach to minimize all the overhead, but this is a reasonable demonstration

Returned Results

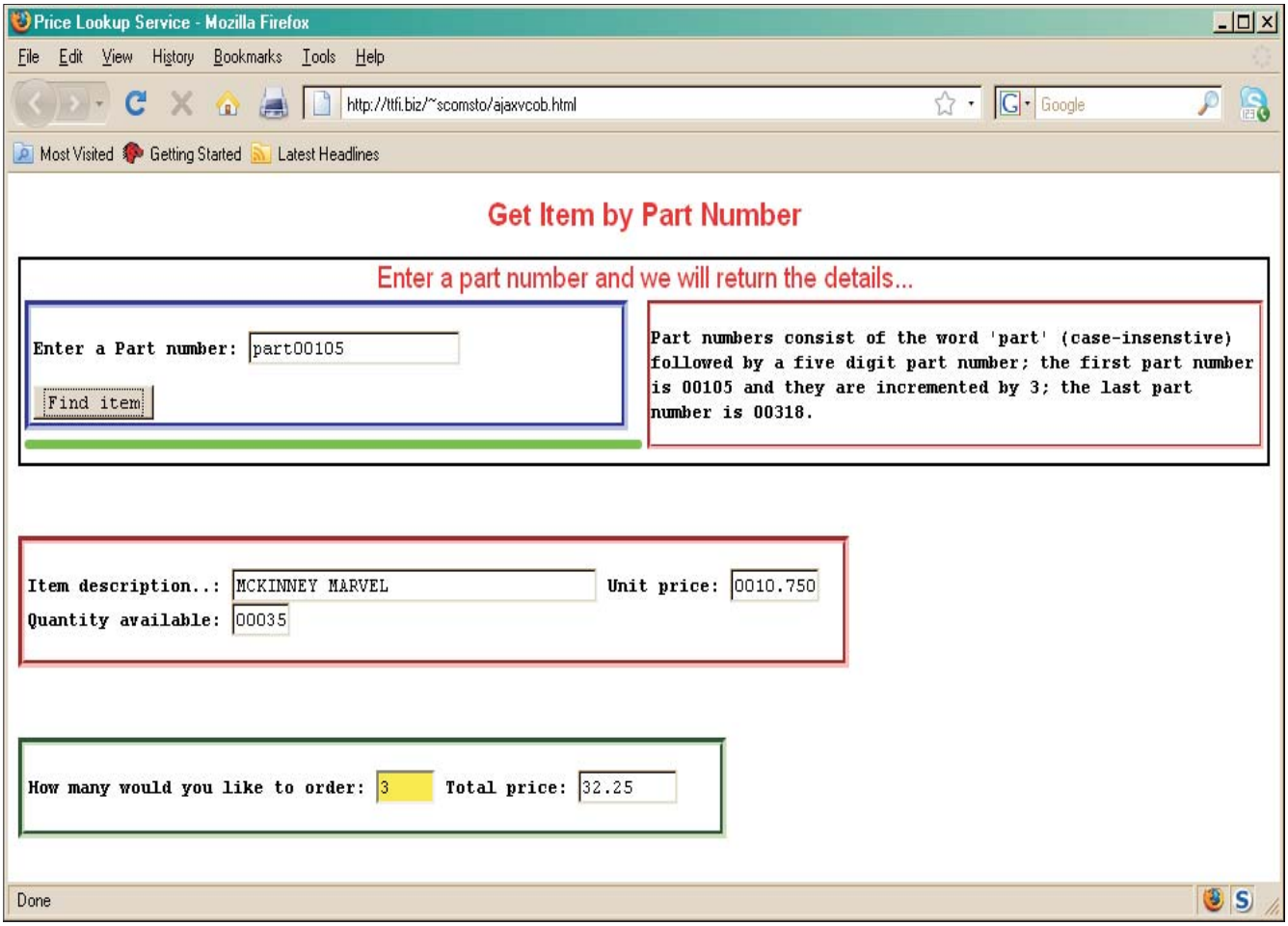
❒ **At this point, you will see the middle section has been filled in and the input controls in the third section have been activated (also, the color has changed for the "quantity" control and that box is given the focus, so the user can simply type in a number-to-order:**



- **Also note the progress bar has completed its journey**

Calculating a Price

- **The "quantity" input control is designed so that if the user enters a number and changes the focus (click, tab, anything like that), the price control will have the calculated value (based on unit price from the CGI times quantity from the user's request), maybe something like this:**



- **So that's the demo, let's look at the pieces...**

The Base Page: ajaxvcob.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Copyright (C) 2009 by Steven H. Comstock -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
<link rel="stylesheet" href="ajax.css" type="text/css" />

<title>Price Lookup Service</title>

<script type="text/javascript">

   var sendReq = new XMLHttpRequest;
   var order_string;
   var first;
   var de_string, unpr_string, qty_str;
   var data_found;
```

The Base Page: ajaxvcob.html, 2

```
function sendReqst()
   {
    document.getElementById('progress').style.width='10%';
    document.getElementById('descr').value = null;
    document.getElementById('unpr').value = null;
    document.getElementById('qty').value = null;
    document.getElementById('ord').value = '';
    document.getElementById('price').value = '';

    order_string =
     document.getElementById('partno').value.toUpperCase();
    sendReq.onreadystatechange=getResponse;
    sendReq.open("GET",
           "/SCOMSTO/ajvcob?order_string="+order_string,true);
    sendReq.send(null);
   }
```

Notes

- **Each time the user clicks the "Find item" button, the sendReqst() script will be run, and this script wil:**
  - ✗ Reset the progress bar
  - ✗ Clear out the data fields
  - ✗ Build "order_string" as the upper case version of whatever value the user entered
    - ➢ Special note: see how the toUpperCase function allows the user not to be concerned with case
    - ➢ Special note: it would probably be a good idea to code some range testing here, in order not to send a request you know will fail
  - ✗ Issue the open() request pointing to the CGI and concatentating the order_string in the CGI style
  - ✗ Send the request

The Base Page: ajaxvcob.html, 3

```
function getResponse()
   {
    if (sendReq.readyState == 1)
       {
        document.getElementById('progress').style.width='25%';
       }
    if (sendReq.readyState == 2)
       {
        document.getElementById('progress').style.width='55%';
       }
    if (sendReq.readyState == 3)
       {
        document.getElementById('progress').style.width='75%';
       }
    if (sendReq.readyState == 4)
       {
        document.getElementById('progress').style.width='100%';
         if (sendReq.status == 200)
            {
             first = sendReq.responseText;
             buildOutput();
            }
         else
            {
             alert(" Request failed with status == " +
                      sendReq.status);
            }
       }
   }
```

Notes

- **This is our event handler; notice that the first three if statements control the width of the progress bar, our** hr **element**
- **Then, when the request is complete (readyState = 4), I invoke our function to handle the response (or issue an error message if the HTTP request was not successful (status not 200)),** buildOutput()

The Base Page: ajaxvcob.html, 4

```
function buildOutput()
  {
   var star=0, lst=0, qty_start = 0;
   data_found = true;
   if (first.slice(1,5) == '<h2>')
     {
      data_found = false;
      document.getElementById('descr').value=
                '** No such part number found **';
      document.getElementById('ord').style.backgroundColor='';
      document.getElementById('ord').setAttribute('disabled',true);
      document.getElementById('price').setAttribute('disabled',true);
      document.getElementById('descr').focus();
     }
   else
     {                /* start of description value   */
      star      = first.indexOf('descr=',0);
                       /* start of unit price value    */
      lst       = first.indexOf('unpr=',0);
                       /* start of qty on hand value   */
      qty_start = first.indexOf('qty=',0);

      de_string=first.slice(star+6,lst - 1);
      document.getElementById('descr').value=de_string;

      unpr_string=first.slice(lst+5,qty_start - 1);
      document.getElementById('unpr').value=unpr_string;

      qty_string=first.slice(qty_start+4);
      document.getElementById('qty').value=qty_string;

      document.getElementById('ord').removeAttribute('disabled');
      document.getElementById('ord').style.backgroundColor='yellow';
      document.getElementById('ord').focus();
     }
  }
```

The Base Page: ajaxvcob.html, 5

Notes for facing page

- **Line breaks are not exactly like in the code, for typographical reasons**
- **The line**

```
if (first.slice(1,5) == '<h2>')
```

is checking to see if the CGI returned an error message

✗ All error messages from the CGI begin with the string <h2>

➢ The slice function grabs data from location 1 up to, but not including, location 5, thus <h2>

- **If an error is found, the description field is filled with the error message, the ord and price fields are disabled, and the focus is put into the description field**
- **If no error is indicated, the code beginning with 'else' parses the response string; recall the string is sent as:**

descr=*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx* **unpr=***9999.999* **qty=***99999*

- **The components are distributed to the description, unit price, and quantity input controls, and the "ord" (how many would you like to order?) field is enabled, color highlighted, and given the focus**

The Base Page: ajaxvcob.html, 6

❒ I round out the script and head elements with the calcPrice function, which is invoked when the "ord" control gets then loses the focus:

```
function calcPrice()
  {
   if (data_found==true)
      {
       var in_price;
       var in_qty;
       var tot_price;

       in_price = Number(document.getElementById('unpr').value)

       in_qty = Number(document.getElementById('ord').value);
       tot_price = in_price * in_qty;
       document.getElementById('price').removeAttribute('disabled');
       document.getElementById('price').value = tot_price.toFixed(2);
      }
  }


</script>

</head>
```

- Notice the use of the Number function to convert to numeric from string with embedded decimal point

- Notice also the use of the toFixed function: this ensures two decimal positions in the result

The Base Page: ajaxvcob.html, 7

❒ Now for the body of the page...

```
<body>

<h1>Get Item by Part Number</h1>
<div class='tablebox'>
  <table  frame='void' >
  <caption>Enter a part number and we will return
           the details... </caption>
    <tr>
    <td width='50%'>
      <form>
        <div class="getdata">
          <p>Enter a Part number:
          <input type="text name="partno" id="partno"  />
          <br />
          </p>
          <input type="button" name="Lookup" id="Lookup"
                 value="Find item" onclick="sendReqst();"/>
        </div>
      </form>

      <hr id='progress' align='left' size='7' noshade width='5%' />

      </td>

    <td align='left' valign='top' >
      <div class='instruction' >
        <p>Part numbers consist of the word 'part'
          (case-insenstive) followed by a five digit part number;
          the first part number is 00105 and they are incremented
          by 3; the last part number is 00318.
        </p>
      </div>
    </td>

    </tr>
  </table>
</div>
```

- ♦ This much describes the first part of the page; take a minute and compare this with the image on page 19)

The Base Page: ajaxvcob.html, 8

❒ The second part of the page, where the results are displayed, is coded as:

```
<br />
<br />
<br />

<div class="feedback">
  <p>
    Item description..: <input readonly id="descr"
                          size="35" />
    Unit price: <input readonly id="unpr" size="8" />
    Quantity available: <input readonly id="qty"
                          size="5" />
  </p>
</div>

<br />
<br />
<br />
```

♦ Notice the three input fields are "readonly"

The Base Page: ajaxvcob.html, 9

❒ The last part of the page, where the user can specify a quantity to order and get back a price, is coded as:

```
<div class="getorder">

  <p>How many would you like to order:
  <input id="ord" size="5" onblur="calcPrice()");
                disabled  />
  Total price:
  <input readonly id="price" size="9" disabled  />
</p>

</div>

</body>
</html>
```

- ♦ Notice that the two input controls here start out disabled; they are enabled in the scripts when the time is ready

❒ The final piece, now, is to look at the invoked COBOL CGI, AJVCOB ...

The COBOL CGI, AJVCOB

- This is an unusual CGI in that it only puts out one header and then some text: no markup

```
 process nodynam xref(short) rent  test(nohook)
* Copyright (C) 2009 by Steven H. Comstock
 Identification division.
 Program-id.  AJVCOB.
*
* COBOL program designed to run as a CGI
*
* - Function is to receive a part number from a form
*   sending back the description and unit price
*

 Environment division.
 Input-Output section.
 File-control.
      Select parts assign to parts
          organization indexed
          access random
          record key is wk-partno
          file status is vsam-stat.

 Data division.
 File section.
 FD  parts.
 01  wk-rec.
      05  wk-partno       pic x(9).
      05  wk-descrption   pic x(30).
      05                  pic x(5).
      05  wk-unit-price   pic s9999v999 packed-decimal.
      05  wk-qty-hand     pic s9(5)     packed-decimal.
      05                  pic x.
      05  wk-qty-order    pic s9999     binary.
      05  wk-reorder      pic s9999     binary.
      05                  pic x(11).
      05  wk-category     pic x(10).
      05                  pic x(23).
```

The COBOL CGI, AJVCOB, 2

```
 Working-storage section.

 01   blank-line pic x value x'15'.

* data items for environment variable work  ----------

 01   Envar-related-variables.
      02  var-name    pic x(13)    value z'QUERY_STRING'.
      02  env-ptr     pointer.
      02  err-ind     redefines env-ptr pic s9(9) binary.
      02  string-len  pic s9(8) binary value 0.
      02  partno      pic x(9)         value spaces.
      02  unpr        pic 9999.999.
      02  quantity    pic 99999.


* items used in VSAM alloation and processing  --------

 01   bpxwdyn             pic  x(8)  value 'BPXWDYN'.

 01   allocate-request.
      02                   pic s9(4)  binary value 50.
      02                   pic x(50)  value
          'alloc fi(parts) dsn(scomsto.train.ksds) shr '.

 01   free-request.
      02                   pic s9(4)  binary value 24.
      02                   pic x(24)  value
           'free fi(parts) '.

 01   vsam-stat  pic 99.

 01   file-info.
      02  open-stat pic x    value '0'.
          88  vsam-open      value '1'.
          88  vsam-closed    value '0'.

 linkage section.
 01   var-value           pic  x(256).
```

The COBOL CGI, AJVCOB, 3

```
 procedure division.
 mainline.
*
*    Emit the front lines of xhtml...
*
       display 'Content-Type: text/plain'
       display blank-line

* following lines extract QUERY_STRING ----

       call 'getenv' using var-name returning env-ptr
       if err-ind = 0
            display '<h2>QUERY_STRING = '
                    '** variable not set **</h2>'
       else
            set  address of var-value to env-ptr
            move 0 to string-len
            inspect var-value tallying string-len for
               characters before initial x'00'
       end-if

* extract the part number sent into wk-partno

       move var-value(14:9) to wk-partno
```

The COBOL CGI, AJVCOB, 4

```
* allocate and open our VSAM file

       call bpxwdyn using allocate-request
       if return-code = 0
             open input parts
             if vsam-stat = 00
                 set vsam-open to true
             else
                 display '<h2>OPEN failed; code: '
                                  vsam-stat '</h2>'
                 goback
             end-if
       else
         display '<h2>Allocation failed; code: '
                         return-code '</h2>'
         goback
       end-if

* Look up the item with the requested part number

       if vsam-open

         read parts
         if vsam-stat = 00
             move wk-unit-price to unpr
             move wk-qty-hand   to quantity

* Construct the response here

             display 'descr=' wk-descrption
             display 'unpr=' unpr
             display 'qty=' quantity

         else
             display
                 '<h2>No record found for item. </h2>'
         end-if
```

The COBOL CGI, AJVCOB, 5

```
* Close and free the file

         close parts
           if vsam-stat = 00
               set vsam-closed to true
               call bpxwdyn using free-request
               if return-code = 0
                  continue
               else
                  display '<h2>Free failed; code: '
                                  return-code '</h2>'
                  goback
               end-if
           else
               display '<h2>CLOSE failed; code: '
                               vsam-stat '</h2>'
               goback
           end-if

       end-if
       goback.
```

❒ All this worked fine on all tested browsers except:

- ♦ Only Firefox displayed the correct color for the horizontal rule used as a progress bar, a small but strange anamoly

The encodeURIComponent Function

- **The part number value passed in this script was a simple string: four alpha characters followed by five numeric digits**
  - **No special characters**
- **Awareness of this becomes important if you need to pass special characters such as spaces, @, #, and so on**
  - **Recall that CGIs expect to receive a string that is "URI encoded"**
    - ***control_name*=*value***, where ***value*** is encoded by:
      - Space characters are replaced by "+"
      - Non-alphanumeric characters are replaced by "%xx" - that is, a per-cent symbol and the two hex characters that compose the ASCII representation of the character
    - The end of each value is an ampersand (&) if there are more variables to come, or a null (x'00') if this is the end of the string
- **If you are passing a complex string to a CGI, you will want to pass it in this style**
  - **Which can be tricky to build**

The encodeURIComponent Function, 2

- **The encodeURIComponent function in ECMAScript (a.k.a. JavaScript) helps immensely**
  - **This function takes a single string expression as input and encodes it in the style of form data**

- **Recall I passed this as the second parameter for my XMLHttpRequest invocation (see p. 24):**

```
"/SCOMSTO/ajvcob?order_string="+order_string
```

  - **If there is a more complex string to pass, you can do it this way:**

```
"cgi_path?descr="+encodeURIComponent(des)
```

  - **Or if you have several pieces, maybe:**

```
"cgi_path?descr="+encodeURIComponent(des)+"&unpr=+unp
```

  - **Or, of course, build up the entire string then pass the result**

```
parm_string="descr="+encodeURIComponent(des)
parm_string+="&unpr="+unp
parm_string+="&supplier="+encodeURIComponent(supplr)
"cgi_path?parm_string
```

- **Note that encodeURIComponent does not encode ~!*()'**

Section Preview

- **Experiment 2 - referencing an existing html fragment**
  - **Loading an Image File**
  - **The ajaxpic.html File**
  - **The innerHTML Property**

Loading an Image File

- I got to thinking how cool it might be to be able to refresh the current page with an image, audio, or other media file
  - But the specs state clearly (and experiment confirmed) that XMLHttpRequest is strictly for text files
- The work around is for the text file to be, say, an img element
- I constructed a one line file, pic1.html, consisting of this line:

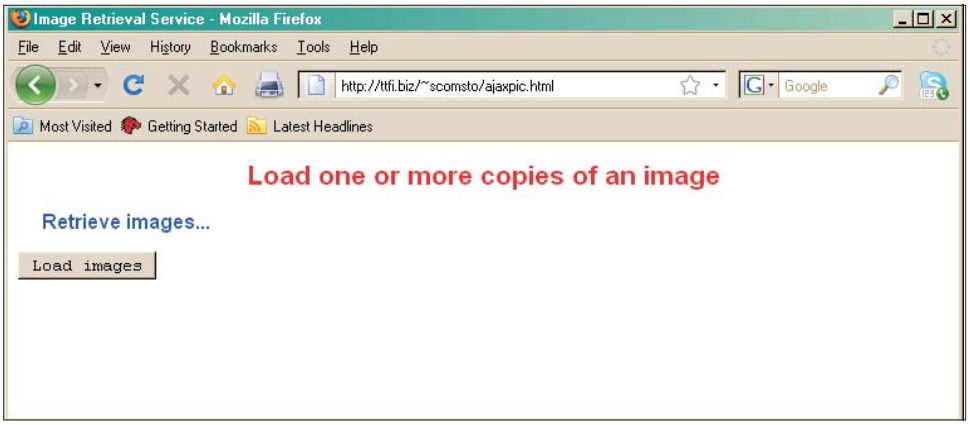
```
<img src="ttflogo2.gif" alt="ttflogo2.gif" >
```

  - The gif file named here is our company logo, and it is in the same directory as the base page file

Loading an Image File, 2

❒ **Next I constructed a simple XHMTL file for testing**

- **The simple page, to start with, looks like this:**



- **When the button is clicked, the image is loaded:**

Loading an Image File, 3

- In fact, each time the button is clicked, and additional copy of the image is loaded:

- The process for this experiment, then, is to design a page with a place to insert the img element

  - I chose a simple paragraph ('p') element ...

The ajaxpic.html File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Copyright (C) 2009 by Steven H. Comstock -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
<link rel="stylesheet" href="ajax.css" type="text/css" />

<title>Image Retrieval Service</title>

<script type="text/javascript">

  var sendReq = new XMLHttpRequest;
  var first;
  var hold_node;
  var source_string;
  var s_strt, s_end;

function sendReqst()
  {
   sendReq.onreadystatechange=getResponse;
   sendReq.open("GET","pic1.html");
   sendReq.send(null);
  }
```

Notes

- **This is very similar to the first part of our first experiment's base page; the differences:**
  - ✗ The title element has different text
  - ✗ The declared variables are a little different
  - ✗ The sendReqst function is simplified, and notice the second operand of the open method invocation is our external file name

The ajaxpic.html File, 2

```
function getResponse()
  {
   if (sendReq.readyState == 4)
      {
       if (sendReq.status == 200)
          {
           first = sendReq.responseText;
           buildOutput();
          }
       else
       {
           alert(" Request failed with status == " +
                 sendReq.status);
          }
      }
  }
```

Notes

- **In this example, I don't use the progress bar code, so I basically need to ensure the result is only processed when the request is done (readyState is 4)**

  - ✗ If the request was successful (a staus of 200), the content of pic1.html is in responseText so insert the element into the base page using the buildOutput function (next page)

  - ✗ If the request failed, I simply issue an error message

The ajaxpic.html File, 3

```
function buildOutput()
   {
     var star=0, lst=0, qty_start = 0;
     hold_node = document.createElement('img');

     s_strt = first.indexOf('src="',0);
     s_strt = s_strt+5;
     s_end  = first.indexOf('"',s_strt);
     source_string = first.slice(s_strt,s_end);

     hold_node.src=source_string;
     hold_node.alt=source_string;

     document.getElementById('pic').appendChild(hold_node);
   }


</script>
</head>
```

Notes

- **The logic of this work is based on work in our course "You and z/OS and the World Wide Web":**
  - ✗ Create an element of type 'img'
  - ✗ Extract the 'src' attribute value from the retrieved file, pic1.html
  - ✗ Set the 'src' and 'alt' attributes of the new element using the value from pic1.html
  - ✗ Append this populated img element as a child to the paragraph element set up as a placeholder on the page

The ajaxpic.html File, 4

❒ The actual body content is miniscule:

```
<body>

<h1>Load one or more copies of an image</h1>
<h2>Retrieve images... </h2>
<input type="button" name="Lookup" id="Lookup"
     value="Load images" onclick="sendReqst();"/>
<p id='pic'></p>

</body>
</html>
```

- ♦ Observe that the paragraph used for the target is empty, and has an id of 'pic' - an ideal place to insert child elements

❒ This worked perfectly in all tested browsers

The innerHTML Property

- There are a number of places in the literature that recommend using a property called innerHTML
  - This property can be applied to a document or an element
  - See, for example: https://developer.mozilla.org/en/DOM/element.innerHTML

- I found this worked, and allowed the entire buildOutput function to be replaced with this code:

```
function buildOutput()
   {
      document.getElementById('pic').innerHTML=first;
   }
```

  - The only difference being, only one image was inserted, no matter how many times you clicked the "Load images" button

The innerHTML Property, 2

- However, there are several places where you are advised against using the feature
  - Mainly because it is not an official part of the standard, and is implemented in different ways on different browsers
- See, for example, these pages on the W3C site where they recommend not using innerHTML

  http://www.w3.org/WAI/GL/WCAG20/WD-WCAG20-SCRIPT-TECHS-20050211/

  http://www.w3.org/TR/2008/WD-WCAG20-TECHS-20081103/SCR21 - Example 1

Section Preview

- **Experiment 3 - referencing an existing XML document**
  - **Handling an XML Document**
  - **The External XML Document: cgioutp.xml**
  - **The newajaxxml.htmlPage**
  - **The New newajaxxml.html Page**
  - **But ... There's A Glitch**
  - **Revision To newajaxsml.html**

Handling an XML Document

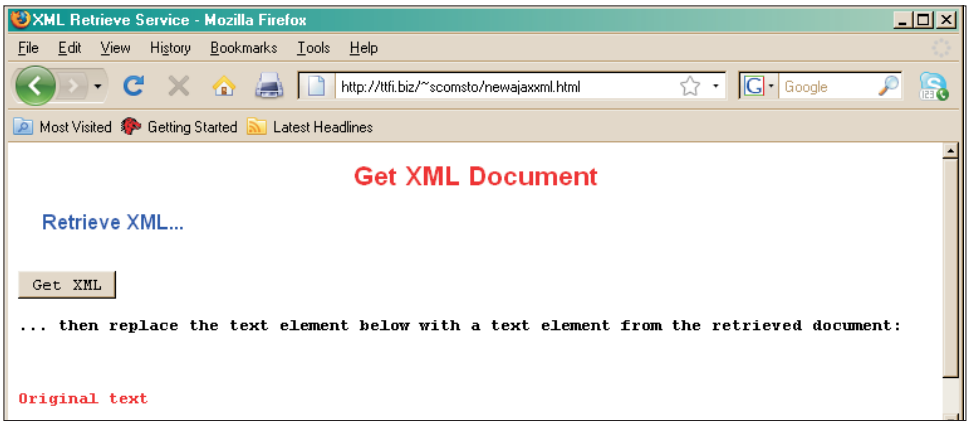
- Well, XML is in the name AJAX and XMLHttpRequest, but I have not used any XML documents at all yet in this document

- For this last experiment, I decided to try and request an entire document that was hanging around from some other work

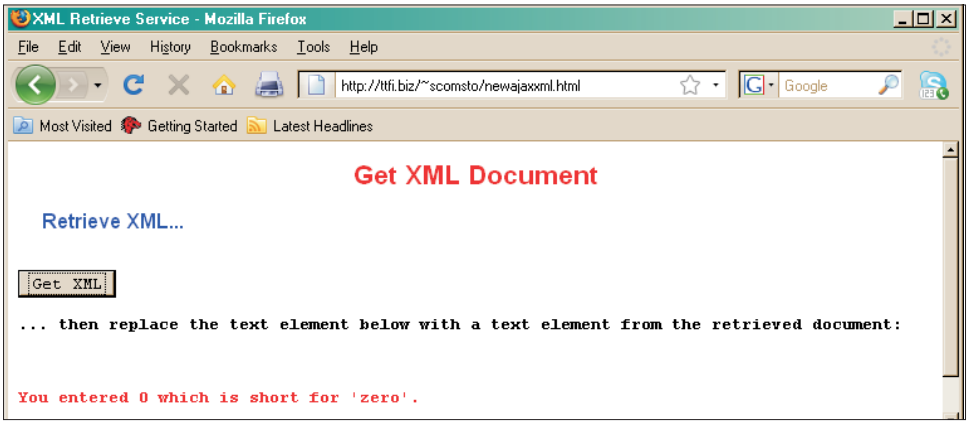
- The idea here is to access this XML document then extract a node from the document and use it to replace a node in the original page

Handling an XML Document, 2

❒ **The original page will look like this:**



♦ **And when the "Get XML" button is clicked, the target XML document is retrieved, and a paragraph element is extracted and used to replace the original paragraph at the bottom of the page:**



The External XML Document: cgioutp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Copyright (C) 2006 by Steven H. Comstock     -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-type" content="text/html;
       charset=UTF-8" />
<title>Output from Assembler CGI on z/OS</title>
</head>
<body  id="body">
<h1 id="h1" >Comin' at ya' from ________</h1>
<p  id="para"> You entered 0 which is short for 'zero'.</p>
<p>
You entered abcdef which, when upper-cased, yields: ABCDEF.
<br />
</p>
</body>
</html>
```

- The paragraph with the id of para is the element I intend to extract, for demonstration purposes

The newajaxxml.html Page

- This is the second XML experiment I created, hence the file name

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Copyright (C) 2011 by Steven H. Comstock -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
<link rel="stylesheet" href="ajax.css" type="text/css" />

<title>XML Retrieve Service</title>

<script type="text/javascript">

   var sendReq = new XMLHttpRequest;
   var first;
   var new_node;
   var old_node;

function sendReqst()
   {
    sendReq.onreadystatechange=getResponse;
    sendReq.open("GET","cgioutp.xml");
    sendReq.send(null);
   }
```

The newajaxxml.html Page, 2

```
      function getResponse()
         {
          if (sendReq.readyState == 4)
             {
              if (sendReq.status == 200)
                 {
                  first = sendReq.responseXML;

                  new_node = first.getElementById('para');
                  old_node = document.getElementById('pic');

document.getElementById('pic').replaceChild(new_node.firstChild,
                                            old_node.firstChild);
                 }
               else
                 {
                  alert(" Request failed with status of " +
                        sendReq.status);
                 }
             }
         }


      </script>
      </head>
```

Notes

- **Note that here I grab responseXML, which contains the whole external document as an XML document object**

- **Even with small type the most important line won't fit on one line here, but the logic is:**

  - ✗ Extract the 'para' node (a paragraph element) from the retrieved document into new_node

  - ✗ Extract the 'pic' node (a paragraph element) from the base document into old_node

  - ✗ Use the replaceChild method to replace old_node's first Child (the text from the paragraph) with new_node's first Child (the text from that paragraph

The newajaxxml.html Page, 3

```
<body>

<h1>Get XML Document</h1>
<h2>Retrieve XML... </h2>

<p><br />

<input type="button" name="Lookup" id="Lookup"
value="Get XML" onclick="sendReqst();"/>

<br />
<br />

... then replace the text element below with a
text element from the retrieved document:

<br />
<br /></p>

<p id="pic" class='red' >Original text</p>
<br />
<br />

</body>
</html>
```

- Notice that the paragraph with id of 'pic' is the second paragraph in the body
  - This example was only successful using Firefox, Chrome, and Opera
  - It failed under Internet Explorer and Safari

The New newajaxxml.html

- **It took a great deal of reading and trial and error to come up with a way to handle XML documents consistently in all the test browsers**

  - **But I finally ended up with this revision to the getResponse function that did the trick:**

```
function getResponse()
   {
    if (sendReq.readyState == 4)
       {
        if (sendReq.status == 200)
           {
            first = sendReq.responseXML;

            new_node =
             first.getElementsByTagName('p')[0].childNodes[0];

            old_node =
             document.getElementsByTagName('p')[1].childNodes[0];

            old_node.nodeValue = new_node.nodeValue;
           }
        else
           {
            alert(" Request failed with status of " +
                  sendReq.status);
           }
       }
   }
```

Notes

- **XML documents seem to require** getElementsByTagName **with a subscript -** getElementById **just didn't cut it**

- **Simlarly, each node had to be modified by childNodes[0] instead of firstChild**

- **Finally, the replaceChild method didn't seem to work, so a simple assignment of one nodeValue to the other got the job done**

But ... There's a Glitch!

- **Although the code shown here works for all the tested browsers, these tests were all done using the HTTP server that comes with z/OS, called the IBM HTTPD Server V5R3M0**

- **When I tested the samples in this document against some ported versions of the Apache server I ran into a strange situation ...**

- **Recall that an XMLHttpRequest returns its value in responseText or responseXML**

  - **What I found out is that whether you get back a text value or an XML document depends on the <u>combination</u> of server and browser you are using:**

    - ✗ The IBM HTTP V5R3M0 server always returned an XML document when I pointed the request at an XML file

    - ✗ But an Apache server I ported to z/OS, and the IBM port of Apache sometimes returned a value in responseText and sometimes in responseXML

    - ✗ So I included code to test if responseXML is null or not and to process the value in responseText or responseXML, using the approrpriate methods for the differing objects

      - ➢ This required adding some additional variables and rewriting the getResponse function to determine which situation I was in ...

Revision To newajaxsml.html

❒ So, a little more flexible version that runs under all three tested servers:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Copyright (C) 2011 by Steven H. Comstock -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
<link rel="stylesheet" href="ajax.css" type="text/css" />

<title>XML Retrieve Service</title>

<script type="text/javascript">

   var sendReq = new XMLHttpRequest;
   var first;
   var n_node;
   var o_node;
   var start_p_content;
   var end_p_content;

function sendReqst()
   {
    sendReq.onreadystatechange=getResponse;
    sendReq.open("GET","cgioutp.xml");
    sendReq.send(null);
   }
```

- Note the new variables

- Also note that I changed the earlier variable names new_node and old_node to n_node and o_node; this was simply for typographical reasons, to better fit on the page in this paper

Revision To newajaxsml.html, 2

```
function getResponse()
 {
 if (sendReq.readyState == 4)
  {
   if (sendReq.status == 200)
    {
     if (sendReq.responseXML == null)    /* response is not XML */
      {
       if (sendReq.responseText == null) /* response is null    */
        {
         alert("Both responseXML and responseText are null");
        }
       else                              /* response is text    */
        {
         first = sendReq.responseText;
         start_p_content = first.indexOf("Yo",1);
         end_p_content = first.indexOf(".",start_p_content);
         n_node = first.substring(start_p_content,end_p_content);
         o_node = document.getElementsByTagName('p')[1].childNodes[0];
         o_node.nodeValue = n_node;
        }
      }
     else                                /* response is XML       */
      {
       first = sendReq.responseXML;
       n_node = first.getElementsByTagName('p')[0].childNodes[0];
       o_node = document.getElementsByTagName('p')[1].childNodes[0];
       o_node.nodeValue = n_node.nodeValue;
      }
    }
   else
    {
     alert(" Request failed with status of " + sendReq.status);
    }
  }
 }
```

- **Note we grab either responseText or responseXML depending on whether responseXML is null or not**

  ✗ Test responseXML first because the standard says that if the user agent (browser) is not a compliant XML parser then the response must be sent back in responseText and responseXML must be null

Revision To newajaxsml.html, 3

```
<body>

<h1>Get XML Document</h1>
<h2>Retrieve XML... </h2>

<p><br />

<input type="button" name="Lookup" id="Lookup"
value="Get XML" onclick="sendReqst();"/>

<br />
<br />

... then replace the text element below with a
text element from the retrieved document:

<br />
<br /></p>

<p id="pic" class='red' >Original text</p>

<p> </p>
<p> </p>

</body>
</html>
```

Notes

- This example was successful under all tested browsers when the server was the IBM V5R3M0 HTTP server
- Under the IBM port of Apache, it only worked with Opera and Internet Explorer but ...

Apache Issues

- **The main issue with Apache and accessing an XML file using XMLHttpRequest seems to be a configuration issue:**
  - **If the XML file is in EBCDIC, I couldn't find a way to get Apache to translate it to ASCII (or UTF-8) before transmitting, so the file arrived at the browser in EBCDIC, which Opera and IE seemed to recognize and handle well**
    - ✗ But Firefox, Chrome, and Safari couldn't handle
  - **If I used an ASCII file to begin with, it was not translated so it arrived as ASCII, and all the browsers tested handled it perfectly**
  - **Meanwhile, the IBM HPPT Server handled the ASCII files fine and it worked for all browsers tested**
  - **At this point in time I have been unsuccessful in finding a combination of configration settings for Apache to support using EBCDIC XML files as the target for an XMLHttpRequest service**

This page intentionally left almost blank.

Section Preview

- Summary and conclusions
  - Coding AJAX Applications
  - Benefits of Using AJAX
  - Drawbacks of Using AJAX

Coding AJAX Applications

- **To use the AJAX style requires careful planning**
  - **Instead of requesting a new page each time, you are requesting an update to a base page**
  - **So you need to know where you will putting updates, and what updates will be allowed**

Benefits of Using AJAX

- If you have many pages with slightly different content, you may gain performance by just modifying a base / common page
  - Instead of generating or retreiving a new page, just update the current page
  - Only load scripts and stylesheets once
- You may use less bandwidth
- Users may perceive performance improvements
- Since all the code is on one page, it is simpler to maintain state information: the variables remaing available

Drawbacks to Using AJAX

- Modified pages are not registered in the browser's history: back button will not back out the last change, but go all the way back to the previously loaded page

- Modified pages cannot be bookmarked, since the changes are not preserved the next time you access the page

- Data you may want to be available (for search engines, say) is not preservable in any meaningful way

- Browsers that do not support JavaScript (or have it turned off) cannot take advantage of AJAX, so you may need to detect this and provide alternative ways of accessing data
  - Thus making an application more complicated, not less

- Currently, AJAX requests may only be for pages from the same domain as the base page
  - Mashup type applications can't work, for example
  - Note that W3C is working on enabling this functionality

- There may be malware susceptibilities that are not yet recognized or defended against

- AJAX applications that do not work in all combinations of servers and browsers may not be good candidates