Asynchronous JavaScript and XML (AJaX)

Object Computing, Inc.
Mark Volkmann
mark@ociweb.com
Topics Covered

• What is AJaX?
• JavaScript Overview
• XMLHttpRequest (XHR)
• Sarissa JavaScript Library
• REST Overview
• Demo Description
• Demo Sequence Diagrams
• Demo REST Server
• Demo XHTML
• Demo JavaScript
• Wrapup
What is AJaX?

• A name given to an existing approach to building dynamic web applications
• Web pages use JavaScript to make asynchronous calls to web-based services that typically return XML
  – allows user to continue interacting with a web page while waiting for data to be returned
  – page can be updated without refreshing browser
  – results in a better user experience
• Uses a JavaScript class called XMLHttpRequest
A Good Acronym?

• **A is for “asynchronous”**
  – requests can be made asynchronously or synchronously
  – both techniques allow web page to be updated without refreshing it
  – anything useful the user can do while processing request?
    • if yes then use asynchronous, otherwise use synchronous

• **J is for “JavaScript”**
  – typically JavaScript is used on the client-side (in the browser)
    • only programming language supported out-of-the-box by most web browsers
  – can use any language on server-side that can accept HTTP requests and return HTTP responses
    • Java servlets, Ruby servlets, CGI scripts, …

• **X is for “XML”**
  – request and response messages can contain XML
    • can easily invoke REST-style services
  – can really contain any text (single text value, delimited text, …)
Uses For AJaX

• **Asynchronous**
  - examples
    • **Google Maps** – [http://maps.google.com](http://maps.google.com)
      – asynchronously loads graphic tiles to support map scrolling
    • **Google Suggest** – [http://www.google.com/suggest](http://www.google.com/suggest)
      – asynchronously updates list of possible topic matches based on what has been typed so far

• **Synchronous**
  - even when there is nothing useful for the user to do after a request is submitted to a server, AJaX can be used to retrieve data and update selected parts of the page without refreshing the entire page
    • better user experience
JavaScript Overview

• A programming language with syntax similar to Java
• Supported by web browsers
  – JavaScript can be downloaded from web servers along with HTML and executed in the browser
• Syntax to use from HTML
  – add `<script>` tag(s) to head section of HTML
  – can embed JavaScript code inside HTML or refer to external JavaScript files
  – embedding
    `<script type="text/javascript"> ... code ... </script>`
  – referring
    `<script type="text/javascript" src="url"></script>`
• JavaScript files cannot include/import others
  – HTML must use a script tag to refer to each needed JavaScript file

these notes use XHTML instead of HTML
XMLHttpRequest

- A JavaScript class supported by most web browsers
- Allows HTTP requests to be sent from JavaScript code
  - to send multiple, concurrent requests, use a different XMLHttpRequest instance for each
- HTTP responses are processed by “handler” functions
- Issue
  - code to create an XMLHttpRequest object differs between browsers
  - can use a JavaScript library such as Sarissa (more detail later) to hide the differences
 XMLHttpRequest Properties
(partial list)

- **readyState**
  - 0 – UNINITIALIZED; open not yet called
  - 1 – LOADING; send not yet called
  - 2 – LOADED; send called, headers and status are available
  - 3 – INTERACTIVE; downloading, responseText only partially set
  - 4 – COMPLETED; finished downloading response

- **responseText**
  - response as text; null if error occurs or ready state < 3

- **responseXML**
  - response as DOM Document object; null if error occurs or ready state < 3

- **status** – integer status code of request

- **statusText** – string status of request
XMLHttpRequest Methods
(partial list)

• Basic methods
  - `open(method, url)` – initializes a new HTTP request
    • `method` can be "GET", "POST", "PUT" or "DELETE"
    • `url` must be an HTTP URL (start with "http://")
  - `send(body)` – sends HTTP request
  - `abort()` – called after `send()` to cancel request

• Header methods
  - `void setRequestHeader(name, value)`
  - `String getResponseHeader(name)`
  - `String getAllResponseHeaders()`
    • returns a string where "header: value" pairs are delimited by carriage returns

Example return value:
Connection: Keep-Alive
Date: Sun, 15 May 2005 23:55:25 GMT
Content-Type: text/xml
Server: WEBrick/1.3.1 (Ruby/1.8.2/2004-12-25)
Content-Length: 1810
Sarissa

• An open source JavaScript library that allows the following to be done in a browser independent way
  – create XMLHttpRequest objects
  – parse XML (synchronously or asynchronously)
  – create XML (using DOM)
  – transform XML with XSLT
  – query XML with XPath

• Download from http://sourceforge.net/projects/sarissa

• Documentation at http://sarissa.sourceforge.net/doc/
Using XMLHttpRequest With Sarissa

• To create an XMLHttpRequest
  ```javascript
  var xhr = new XMLHttpRequest();
  ```

• To send synchronous GET request and obtain response
  ```javascript
  xhr.open("GET", url, false); // false for sync
  var body = null; // wouldn’t be null for a POST
  xhr.send(body);
  var domDoc = xhr.responseXML;
  var xmlString = Sarissa.serialize(domDoc);
  ```

• To send asynchronous GET request
  ```javascript
  xhr.open("GET", url, true); // true for async
  var body = null; // wouldn’t be null for a POST
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      var domDoc = xhr.responseXML;
      var xmlString = Sarissa.serialize(domDoc);
    }
  }
  xhr.send(body);
  ```

Sarissa.serialize gets a string representation of an DOM node
Using XMLHttpRequest With Sarissa (Cont’d)

• To set a request header
  
  ```javascript
  xhr.setRequestHeader("name", "value");
  ```

• To get a response header
  
  ```javascript
  var value = xhr.getResponseHeader("name");
  ```
REST Overview

- Stands for **RE**presentational **S**tate **T**ransfer
- Main ideas
  - a software component requests a “**resource**” from a service
    - by supplying a resource identifier and a desired media type
  - a “**representation**” of the resource is returned
    - a sequence of bytes and metadata to describe it
      - metadata is name-value pairs (can use HTTP headers)
  - obtaining this representation causes the software component to “**transfer**” to a new “**state**”
REST Overview (Cont’d)

• REST is an architectural style, not a standard or an API
  – but can use existing standards including URLs, HTTP and XML
  – can be implemented in many ways (such as Java or Ruby servlets)
  – used to build distributed applications such as Web apps. and Web services

• Good sources for further reading
  – “Building Web Services the REST Way” by Roger L. Costello
    • http://www.xfront.com/REST-Web-Services.html
  – Roy Fielding’s 2000 dissertation (chapter 5)
    • http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
  – RESTwiki - http://rest.blueoxen.net/cgi-bin/wiki.pl
  – REST mailing list - http://groups.yahoo.com/group/rest-discuss/
REST Resources and Identifiers

• **What is a REST resource?**
  – a specific, retrievable thing, not an abstract concept
  – for example, instead of having a “car” resource with representations like “photo” and “sales report”, those are the resources
    • **car photo** from a specific view (front, side and rear) with JPEG representations
    • **car sales report** for a specific month/year with PDF and XML representations

• **What are good resource identifiers?**

  http://host:port/webapp/carPhoto
  ?make=BMW&model=Z3&year=2001&view=front
  http://host:port/webapp/carSalesReport
  ?make=BMW&model=Z3&year=2001&salesYear=2004&salesMonth=4

“Think of RESTful applications to consist of objects (resources) that all have the same API (PUT, DELETE, GET, POST, etc). For a component of the application to invoke a method on an object, it issues an HTTP request.”
*from a post on the rest-discuss by Jan Algermissen*

An **underlying goal** is to make as many things as possible retrievable by an HTTP GET request. This enables **browser-based testing**.
Demo Description

• **Music collection search**
  – MySQL database is populated off-line from an iTunes XML file
  – web page contains
    • text field to select artist name
      – suggests completions like Google Suggest
      – database columns include id and name
    • list of CDs by that artist
      – updated asynchronously when an artist name is entered
      – database columns include id, title and year
    • table of track data for that CD
      – updated asynchronously when CD selection changes
      – database columns include id, track number, name, time and rating
  – requests and responses follow REST style
### Music Collection

<table>
<thead>
<tr>
<th>Artist</th>
<th>CDs</th>
<th>Tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Björk</td>
<td>Homogenic</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Post</td>
<td>1 Army Of Me</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Hyper Ballad</td>
</tr>
<tr>
<td></td>
<td>Debut</td>
<td>3 The Modern Things</td>
</tr>
<tr>
<td></td>
<td>Voepportino</td>
<td>4 It's Oh So Quiet</td>
</tr>
<tr>
<td></td>
<td>Selmasongs</td>
<td>5 Enjoy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 You've Been Flirting Again</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 Isobel</td>
</tr>
<tr>
<td>Baker, Anita</td>
<td></td>
<td>8 Possibly Maybe</td>
</tr>
<tr>
<td>Bella Wolf</td>
<td></td>
<td>9 I Miss You</td>
</tr>
<tr>
<td>Belly</td>
<td></td>
<td>10 Cover Me</td>
</tr>
<tr>
<td>Bic Runga</td>
<td></td>
<td>11 Headphones</td>
</tr>
<tr>
<td>Björk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob Geldof</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bode, Erin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bonham, Tracy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breeders, The</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brooks, Meredith</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reset**
Demo Pieces
(we’ll focus on boxes with bold text)
Getting Artists Whose Names Begin With *prefix*

- Request
  

- Response

```xml
<artists>
  <artist id="141" href="http://localhost:2000/music/artist?id=141">
    Cocteau Twins
  </artist>
  <artist id="72" href="http://localhost:2000/music/artist?id=72">
    Cole, Holly
  </artist>
  <artist id="80" href="http://localhost:2000/music/artist?id=80">
    Cole, Paula
  </artist>
  <artist id="111" href="http://localhost:2000/music/artist?id=111">
    Collins, Phil
  </artist>
  <artist id="48" href="http://localhost:2000/music/artist?id=48">
    Colvin, Shawn
  </artist>
    Counting Crows
  </artist>
  <artist id="54" href="http://localhost:2000/music/artist?id=54">
    Cowboy Junkies
  </artist>
</artists>
```
Getting Artist Information

• Request

• Response
  <artist id="97">
    <name>Apple, Fiona</name>
    <cd artistId="97" id="163">
      <title>When The Pawn...</title>
      <track rating="3" id="767" cdId="163">On The Bound</track>
      <track rating="3" id="768" cdId="163">To Your Love</track>
      ...
    </cd>
    <cd artistId="97" id="164">
      <title>Tidal</title>
      <track rating="4" id="777" cdId="164">Sleep To Dream</track>
      <track rating="4" id="778" cdId="164">Sullen Girl</track>
      ...
    </cd>
  </artist>

Request

Response
  <artist id="97">
    <name>Apple, Fiona</name>
  </artist>

  without “deep”
Getting CD Information

• Request

• Response
  <cd artistId="97" id="164">
    <title>Tidal</title>
    <track rating="4" id="777" cdId="164">Sleep To Dream</track>
    <track rating="4" id="778" cdId="164">Sullen Girl</track>
    ...
  </cd>

Request
Response
  <cd artistId="97" id="164">
    <title>Tidal</title>
    ...
  </cd>

  without “deep”
Getting Track Information

• Request

• Response
  <track rating="4" id="777" cdId="164">Sleep To Dream</track>
artistInput onkeydown & onkeyup
Event Handling

continued on next diagram
handleArtists Function
artistSelect and cdSelect onchange Event Handling
MusicServer.rb

• Implemented in Ruby
• Uses WEBrick
  – http://www.webrick.org
  – “a Ruby library program to build HTTP servers”
  – “a standard library since Ruby-1.8.0”
#!/usr/bin/ruby

require 'mysql'
require 'rexml/document'
require 'webrick'

include REXML
include WEBrick

# Add to_s method to REXML Element class.
class Element
  def to_s
    s = ''; write(s); s
  end
end
SERVLET_HOST = 'localhost'
SERVLET_PORT = 2000
SERVLET_NAME = 'music'

class MusicServlet < HTTPServlet::AbstractServlet

    DATABASE = 'music'
    DB_HOST = 'localhost'
    DB_USERNAME = 'root'
    DB_PASSWORD = ''

    # A new servlet instance is created to service each request
    # so currently a new database connection is being created for each.
    # TODO: Consider using a pool of database connections.
    # TODO: See http://segment7.net/projects/ruby/WEBbrick/servlets.html
    def initialize(server)
      super(server)
      @conn = Mysql.new(DB_HOST, DB_USERNAME, DB_PASSWORD, DATABASE)
    end

    def get_resource_url(type, id)
      "http://#{SERVLET_HOST}:#{SERVLET_PORT}/#{SERVLET_NAME}/#{type}?id=#{id}"
    end
def do_GET(req, res)
    resource_type = req.path_info[1..-1] # remove first character
    resource_id = req.query['id']
    starts = req.query['starts']
    @deep = req.query['deep']

    res['Content-Type'] = 'text/xml'
    res.body = case resource_type
        when 'artist'
            if resource_id
                get_artist(resource_id).to_s
            else
                get_all_artists(starts).to_s
            end
        when 'cd'
            get_cd(resource_id).to_s
        when 'track'
            get_track(resource_id).to_s
        else
            "unsupported resource type #{resource_type}" end
    end
end
def get_all_artists(starts)
    sql = "select * from artists"
    sql += " where name like '#{starts}%'' if starts
    sql += " order by name"
    rs = @conn.query(sql)

    artists = Element.new('artists') # root element

    rs.each_hash do |row|
        artist = Element.new('artist', artists) # add artist element to root element
        id = row['id']
        artist.add_attribute('id', id)
        artist.add_attribute('href', get_resource_url('artist', id))
        artist.add_text(row['name'])
    end

    artists
end
def get_artist(artist_id)
    sql = "select * from artists where id='#{artist_id}"
    rs = @conn.query(sql)
    return "no artist with id #{artist_id} found" if rs.num_rows == 0

    row = rs.fetch_hash
    artist = Element.new('artist') # root element
    artist.add_attribute('id', artist_id)
    name = Element.new('name', artist) # add name element to root element
    name.add_text(row['name'])

    sql = "select * from cds where artistId=#{artist_id}"
    rs = @conn.query(sql)
    rs.each_hash do |row|
        cd_id = row['id']
        cd = if @deep
            artist.add_element(get_cd(cd_id)) # add cd element to artist element
        else
            Element.new('cd', artist) # add cd element to artist element
        end
        cd.add_attribute('id', cd_id)
        cd.add_attribute('href', get_resource_url('cd', cd_id)) if not @deep
    end

    artist
end
def get_cd(cd_id)
    sql = "select * from cds where id='#{cd_id}'"
    rs = @conn.query(sql)
    return "no cd with id #{cd_id} found" if rs.num_rows == 0

    row = rs.fetch_hash
    cd = Element.new('cd') # root element
    cd.add_attribute('id', cd_id)
    cd.add_attribute('artistId', row['artistId'])
    title = Element.new('title', cd) # add title element to root element
    title.add_text(row['title'])

    sql = "select * from tracks where cdId=#{cd_id}"  # select all tracks for the CD
    rs = @conn.query(sql)
    rs.each_hash do |row|
        track_id = row['id']  # get track ID
        track = if @deep
                   cd.add_element(get_track(track_id)) # add track element to CD element
                else
                Element.new('track', cd) # add track element to CD element
                end
        track.add_attribute('href', get_resource_url('track', track_id)) if not @deep
    end

    cd
end
def get_track(track_id)
    sql = "select * from tracks where id=#{track_id}".as_quoted
    rs = @conn.query(sql)
    return "no track with id #{track_id} found" if rs.num_rows == 0

    row = rs.fetch_hash
    track = Element.new('track')  # root element
    track.add_attribute('id', track_id)
    track.add_attribute('cdId', row['cdId'])
    track.add_attribute('rating', row['rating'])
    track.add_text(row['name'])

    track
end

end # class MusicServlet
MusicServer.rb (Cont’d)

```ruby
# Create WEBrick server, mount the servlet and start the server.
s = HTTPServer.new(:Port=>SERVLET_PORT)
s.mount("/#{SERVLET_NAME}", MusicServlet)
trap('INT') { s.shutdown } # shutdown on Ctrl-C
s.start
```
MusicCollection.xhtml

<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Music Collection</title>

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

    <!-- TODO: Why can't I use shortcut element termination here? -->
    <script type="text/javascript" src="../../sarissa/sarissa.js"></script>
    <script type="text/javascript" src="DHTMLUtil.js"></script>
    <script type="text/javascript" src="StringUtil.js"></script>
    <script type="text/javascript" src="MusicCollection.js"></script>
  </head>
  <body>
    <h1>Music Collection</h1>
  </body>
</html>
<form id="myForm" action="">
  <table>
    <tr>
      <th id="artistHeader">Artist</th>
      <th id="cdHeader">CDs</th>
      <th id="trackHeader">Tracks</th>
    </tr>
    <tr>
      <td valign="top">
        <input type="text" id="artistInput" tabindex="1"
          onkeydown="artistKeydown(event, this)"
          onkeyup="artistKeyup(event, this)" />
      </td>
      <td valign="top" rowspan="2">
        <select id="cdSelect" tabindex="3" size="12"
          onchange="cdSelected(this)">
          <option></option> <!-- XHTML requires at least one option --></select>
      </td>
    </tr>
  </table>
</form>
<td valign="top" rowspan="2">

<table id="trackTable">
<tr>
<th id="trackNumber">#</th>
<th id="trackName">Name</th>
<th id="trackRating">Rating</th>
</tr>
</table>
</td>
</tr>
<tr>
<td id="artistSelectTD">
<select id="artistSelect" tabindex="2" size="10" onchange="artistSelected(this)"
  >
  <option><!-- XHTML requires at least one option --></option>
</select>
</td>
</tr>
</table>
<!-- for debugging -->
<!--p><textarea id="log" rows="20" cols="80"></textarea></p-->  
<p><input type="reset" /></p>
</form>
</body>
</html>
// This contains utility functions make working with DHTML easier.

// Removes all the options from a given select component.
function clearSelect(select) {
    while (select.length > 0) {
        select.remove(0);
    }
}

// Gets the text inside a given DOM element.
// TODO: This should really concatenate the values
// of all text nodes inside the element.
function getText(element) {
    return element.firstChild.nodeValue;
}

// Logs a message to a text area with an id of "log"
// for debugging purposes.
function log(message) {
    document.getElementById("log").value += message + "\n";
}
DHTMLUtil.js (Cont’d)

// Sends an asynchronous HTTP request to a given URL
// whose response will be sent to a given handler.
function send(xhr, url, handler) {
    async = true;
    xhr.onreadystatechange = handler;
    //log("send: opening " + url);
    xhr.open("GET", url, async);
    //log("send: sending to " + url);
    body = null;
    xhr.send(body);
}
// Keycodes used by event handling functions.
var backspaceKeycode = 8;
var ctrlKeycode = 17;
var downArrowKeycode = 40;
var shiftKeycode = 16;

// Base URL of asynchronous HTTP requests.
var baseURL = "http://localhost:2000/music/";

// Keeps track of whether the Ctrl key is currently down.
var ctrlKeyDown = false;

// The characters of the artist name that the user typed.
var lastArtistPrefix = ""

// Used to send asynchronous HTTP requests.
var xhr = new XMLHttpRequest(); // from Sarissa
MusicCollection.js (Cont’d)

// Handles keydown events in the artist input field.
function artistKeydown(event, component) {
    if (event.keyCode == ctrlKeycode) ctrlKeyDown = true;
    if (event.keyCode == downArrowKeycode) {
        // Move focus from artistInput to artistSelect.
        document.getElementById("artistSelect").focus();
    }
}

// Handles keyup events in the artist input field.
function artistKeyup(event, component) {
    if (!ctrlKeyDown) getArtists(event, component);
    if (event.keyCode == ctrlKeycode) ctrlKeyDown = false;
}
// Handles selections of artists in the artist select component.
function artistSelected(component) {
    index = component.selectedIndex;
    value = component.options[index].text;
    document.getElementById("artistInput").value = value;
    getCDs();  // asynchronously
}

// Handles selections of CDs in the CD select component.
function cdSelected(component) {
    index = component.selectedIndex;
    cdId = component.options[index].value;
    getTracks(cdId);  // asynchronously
}
// Sends an asynchronous request to obtain
// a list of artists whose name begins with
// the prefix entered in a text input component.
function getArtists(event, component) {
    if (event.keyCode == shiftKeyCode) return;

    if (event.keyCode == backspaceKeyCode) {
        artistPrefix = lastArtistPrefix.substring
        (0, lastArtistPrefix.length - 1);
    } else {
        artistPrefix = ltrim(component.value); // in StringUtil.js
    }
    lastArtistPrefix = artistPrefix

    if (artistPrefix.length == 0) {
        component.value = "";
        clearSelect(document.getElementById("artistSelect"));
    } else {
        url = baseURL + "artist?starts=" + artistPrefix;
        send(xhr, url, handleArtists);
    }
}
// Sends an asynchronous request to obtain
// a list of CDs by the artist selected in a select component.
function getCDs() {
    select = document.getElementById("artistSelect");
    index = select.selectedIndex;
    option = select.options[index];
    artistId = option.value
    url = baseURL + "artist?id=" + artistId + "&deep";
    send(xhr, url, handleCDs);
}

// Sends an asynchronous request to obtain
// a list of tracks on a CD selected in a select component.
function getTracks(cdId) {
    url = baseURL + "cd?id=" + cdId + "&deep";
    send(xhr, url, handleTracks);
}
// Handles the response from asynchronous requests
// for information about artists
// whose name begins with a given prefix.
function handleArtists() {
  if (xhr.readyState == 4) {
    doc = xhr.responseXML;
    //log("handleArtists: xml = " + Sarissa.serialize(doc));
    if (doc.documentElement == null) {
      alert("Is the server running?");
      return;
    }

    doc.setProperty("SelectionLanguage", "XPath");
    id = doc.selectSingleNode("/"); // from Sarissa
    nodes = doc.selectNodes("/artists/artist"); // from Sarissa

    artistSelect = document.getElementById("artistSelect");
    clearSelect(artistSelect);

    if (nodes.length == 0) return;
}
MusicCollection.js (Cont’d)

// Add an option to artistSelect for each matching artist.
for (i = 0; i < nodes.length; i++) {
    artist = nodes[i];
    name = getText(artist);
    id = artist.getAttribute('id')
    option = new Option(name, id, false, i == 0);
    artistSelect.add(option);
}

// Set artist text field to first choice.
input = document.getElementById("artistInput");
firstArtistName = getText(nodes[0]);
input.value = firstArtistName;

// Highlight suffix supplied by search.
enteredLength = lastArtistPrefix.length;
totalLength = firstArtistName.length
range = input.createTextRange();
range.moveStart("character", enteredLength);
range.moveEnd("character", totalLength);
range.select();

getCDs();
}
// Handles the response from asynchronous requests
// for information about CDs by an artist.
function handleCDs() {
    if (xhr.readyState == 4) {
        doc = xhr.responseXML;
        // log("handleCDs: xml = " + Sarissa.serialize(doc));

        doc.setProperty("SelectionLanguage", "XPath");
        id = doc.selectSingleNode("/"); // from Sarissa
        nodes = doc.selectNodes("/artist/cd"); // from Sarissa

        select = document.getElementById("cdSelect");
        clearSelect(select);
firstId = 0;

// Add an option to cdSelect for each CD.
for (i = 0; i < nodes.length; i++) {
    cd = nodes[i];
    title = getText(cd.selectSingleNode("title")); // from Sarissa
    id = cd.getAttribute('id');
    if (i == 0) firstId = id;
    option = new Option(title, id, i == 0);
    select.add(option);
}

select.selectedIndex = 0;
getTracks(firstId);
}
// Handles the response from asynchronous requests
// for information about tracks on a CD.
function handleTracks() {
  if (xhr.readyState == 4) {
    doc = xhr.responseXML;
    //log("handleTracks: xml = " + Sarissa.serialize(doc));

    doc.setProperty("SelectionLanguage", "XPath");
    id = doc.selectSingleNode("/"); // from Sarissa
    nodes = doc.selectNodes("/cd/track"); // from Sarissa

    table = document.getElementById("trackTable");

    // Delete all the table rows except the header row.
    rowCount = table.rows.length;
    for (i = rowCount - 1; i > 0; i--) {
      table.deleteRow(i);
    }
}
// Add a row to trackTable for each track.
for (i = 0; i < nodes.length; i++) {
    track = nodes[i];
    name = getText(track);
    id = track.getAttribute('id');
    rating = track.getAttribute('rating');

    row = table.insertRow(i + 1);
    row.bgColor = "white";

    cell = row.insertCell(0); // track number
    cell.align = "right"
    cell.innerHTML = i + 1;

    cell = row.insertCell(1); // track name
    cell.innerHTML = name;
    if (rating >= 4) cell.className = "favorite";

    cell = row.insertCell(2); // track rating
    cell.align = "center"
    cell.innerHTML = rating;
}
Wrap Up

• **Summary**
  – don’t have to refresh the browser page in order to display new data from the server
  – get data asynchronously with XMLHttpRequest

• **ToDos**
  – get JavaScript code to work in browsers other than IE6
  – test performance with REST server and web server running on different machines than browser
  – could improve performance by caching REST responses in client-side JavaScript

• **Questions?**