

# Asynchronous JavaScript and XML (AJaX)

Object Computing, Inc.

Mark Volkmann

mark@ociweb.com



1

AJaX

## 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



2

AJaX

## 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 web page while waiting for data to be returned
  - page can be updated without refreshing browser
  - results in a better user experience
  - there are AJaX libraries that reduce the amount of JavaScript code that must be written
- Uses a JavaScript class called XMLHttpRequest



3

AJaX

## 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, ...)



4

AJaX

## Uses For AJaX

- **Asynchronous**

- examples

- Google Maps – <http://maps.google.com>
      - asynchronously loads graphic tiles to support map scrolling
    - Google 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



5

AJaX

## 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>`

these notes use XHTML instead of HTML

The XHTML DTD declaration for the `script` tag says `<!ELEMENT script (#PCDATA)>`, and the XHTML specs says “Given an empty instance of an element whose content model is not EMPTY (for example, an empty title or paragraph) do not use the minimized form (e.g. use `<p>` `</p>` and not `<p />`).



6

AJaX

## JavaScript Overview (Cont'd)

- **JavaScript files cannot include/import others**
  - HTML must use a script tag to refer to each needed JavaScript file



7

AJaX

## 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**
  - in client-side JavaScript
- **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



8

AJaX

## XMLHttpRequest Properties

(partial list)

- **readyState**
  - 0 = UNINITIALIZED; open not yet called
  - 1 = LOADING; send for request not yet called
  - 2 = LOADED; send called, headers and status are available
  - 3 = INTERACTIVE; downloading response,  
    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
- **statusText** – string status

this is a property of  
many JavaScript objects

usually wait for  
`xhr.readyState == 4`



9

AJAX

## XMLHttpRequest Methods

(partial list)

- **Basic methods**
  - `open(method, url[, async])` – initializes a new HTTP request
    - `method` can be "GET", "POST", "PUT" or "DELETE"
    - `url` must be an HTTP URL (start with "http://")
    - `async` is a boolean indicating whether request should be sent asynchronously
      - defaults to true
  - `send(body)` – sends HTTP request
  - `abort()` – called after `send()` to cancel request

**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



10

AJAX

## Sarissa

- An open source JavaScript library that allows the following to be done in a browser independent way
  - create XMLHttpRequest objects (`sarissa.js`)
  - parse XML using DOM (synchronous) or SAX (async.) style (`sarissa.js`)
  - create XML using DOM (`sarissa.js`)
  - transform XML using XSLT (`sarissa_ieemu_xslt.js`)
  - query XML using XPath (`sarissa_ieemu_xpath.js`)
- Download from <http://sourceforge.net/projects/sarissa>
- Documentation at <http://sarissa.sourceforge.net/doc/>



11

AJax

## Using XMLHttpRequest With Sarissa

- To create an XMLHttpRequest

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

- To send synchronous GET request and obtain response

```
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);
```

Sarissa.serialize  
gets a string representation  
of an DOM node;  
mainly used for debugging

- To send asynchronous GET request

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

function is called every time  
readyState value changes;  
can set onreadystatechange  
to the name of a function  
defined elsewhere



12

AJax

## Using XMLHttpRequest With Sarissa (Cont'd)

- To set a request header

```
xhr.setRequestHeader("name", "value");
```

- To get a response header

```
var value = xhr.getResponseHeader("name");
```



13

AJAX

## REST Overview

- Stands for REpresentational State Transfer
- 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**”



14

AJAX

## 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/>



15

AJaX

## 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

“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*

### • What are good resource identifiers?

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

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



16

AJaX

## Demo Description

- Music collection search
  - MySQL database is populated off-line from an iTunes XML file
  - web page contains
    - text field to enter an artist name
      - suggests completions like Google Suggest
      - database columns include id and name
    - list of artists whose name matches what has been typed so far
      - update asynchronously during typing
    - list of CDs by the selected artist
      - updated asynchronously when an artist name is entered or selected
      - database columns include id, title and year
    - table of track data for selected CD
      - updated asynchronously when CD selection changes
      - database columns include id, track number, name, time and rating
  - requests and responses follow REST style



17

AJaX

## Demo Screenshot

A screenshot of a Microsoft Internet Explorer window displaying the "Music Collection" application. The window title is "Music Collection - Microsoft Internet Explorer". The address bar shows the URL "C:\AJaX\MusicCollection\webpage\MusicCollection.xhtml".

The main content area displays three tables:

- Artist:** A list of artists including Björk, Baker, Anita, Bella Wolf, Belly, Bic Runga, Bjork, Bob Geldof, Bode, Erin, Bonham, Tracy, Breeders, The, Brooks, Meredith.
- CDs:** A list of CDs for Björk: Homogenic, Post, Debut, Vespertine, Selmasongs.
- Tracks:** A table of tracks with columns #, Name, and Rating. The table includes rows 1 through 11. Track names are bolded if their rating is greater than or equal to 4.

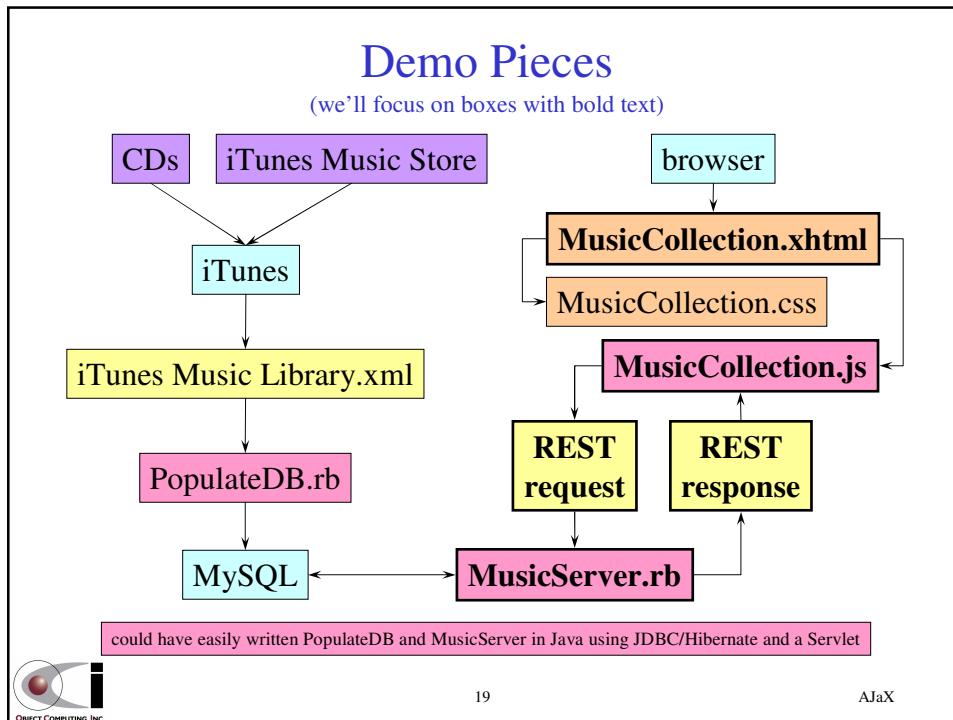
A yellow callout box points to the "Hyper-Ballad" entry in the tracks table, with the text "track names are bold if rating >= 4".

At the bottom of the page are buttons for "Reset" and "Done".



18

AJaX



## Getting Artist Information

- Request

```
http://localhost:2000/music/artist?id=97&deep
```

- 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  
http://localhost:2000/music/artist?id=97  
Response  
<artist id="97">  
 <name>Apple, Fiona</name>  
 <cd href="http://localhost:2000/music/cd?id=163" id="163" />  
 <cd href="http://localhost:2000/music/cd?id=164" id="164" />  
</artist>

without "deep"

21

AJaX

## Getting CD Information

- Request

```
http://localhost:2000/music/cd?id=164&deep
```

- 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  
http://localhost:2000/music/cd?id=164  
Response  
<cd artistId="97" id="164">  
 <title>Tidal</title>  
 <track href="http://localhost:2000/music/track?id=777" />  
 <track href="http://localhost:2000/music/track?id=778" />  
 ...
</cd>

without "deep"

22

AJaX

## Getting Track Information

- Request

`http://localhost:2000/music/track?id=777`

- Response

```
<track rating="4" id="777" cdId="164">Sleep To Dream</track>
```

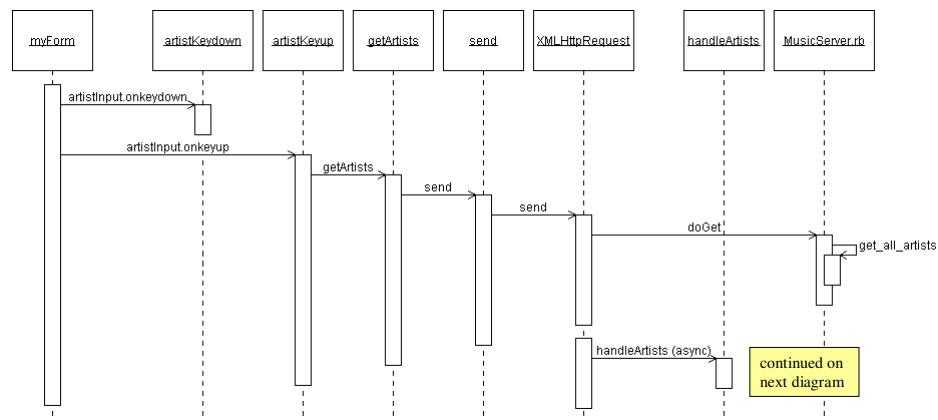


23

AJAX

## artistInput onkeydown & onkeyup Event Handling

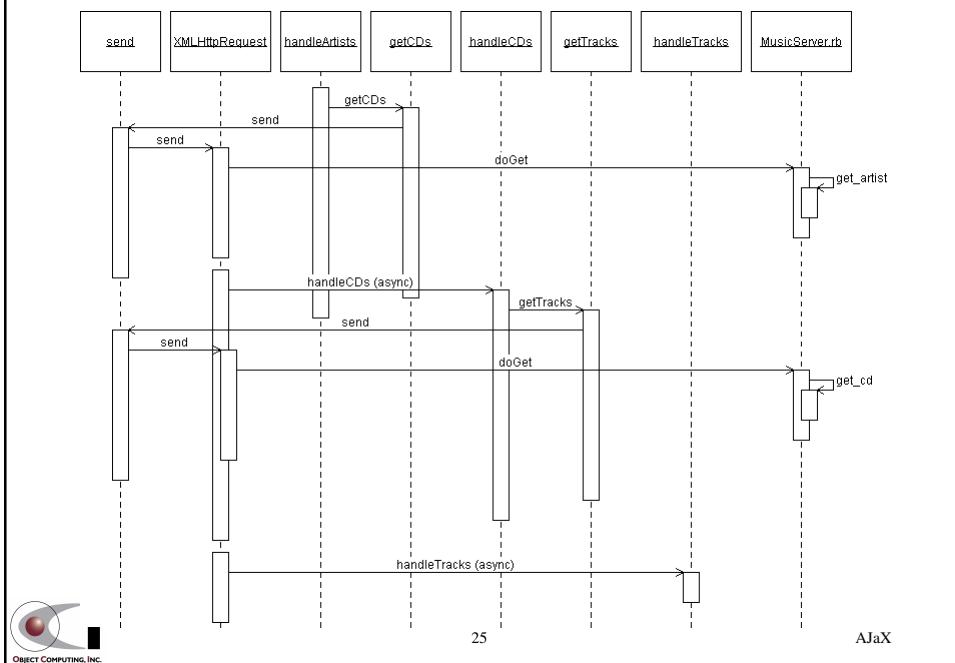
WARNING: This is an unusual use of a sequence diagram where many of the boxes are JavaScript functions, not objects.



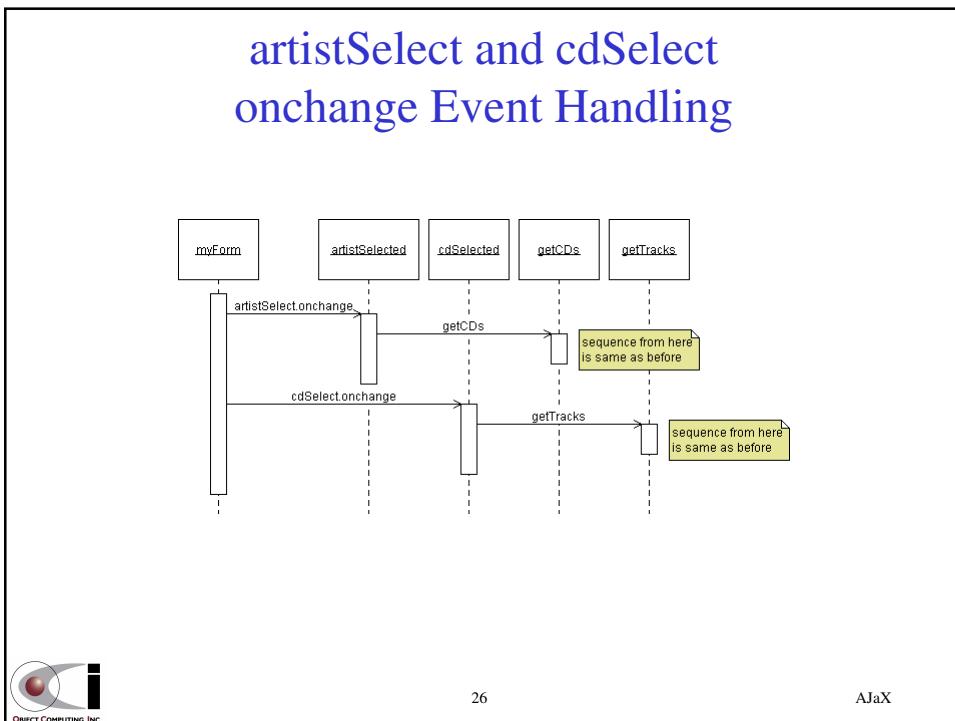
24

AJAX

## 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”



27

AJaX

## MusicServer.rb (Cont'd)

```
#!/usr/bin/ruby

require '../environment.rb' # setup for using Active Record to query database
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
```



28

AJaX

## MusicServer.rb (Cont'd)

```
SERVLET_HOST = 'localhost'  
SERVLET_PORT = 2000  
SERVLET_NAME = 'music'  
  
class MusicServlet < HTTPServlet::AbstractServlet  
  
  # A new servlet instance is created to service each request.  
  def initialize(server)  
    super(server)  
  end  
  
  def get_resource_url(type, id)  
    "http://#{SERVLET_HOST}:#{SERVLET_PORT}/#{SERVLET_NAME}/#{type}?id=#{id}"  
  end
```



29

AJaX

## MusicServer.rb (Cont'd)

```
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 and resource_id.size > 0  
        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
```

invoking to\_s method we added to REXML Element class



30

AJaX

## MusicServer.rb (Cont'd)

```
def get_all_artists(starts)
  artists_element = Element.new('artists')

  artists = Artist.starts_with(starts)

  artists.each do |artist|
    artist_element = Element.new('artist', artists_element)
    artist_element.add_attribute('id', artist.id)
    artist_element.add_attribute(
      'href', get_resource_url('artist', artist.id))
    artist_element.add_text(artist.name)
  end

  artists_element
end
```



31

AJaX

## MusicServer.rb (Cont'd)

```
def get_artist(artist_id)
  artist = Artist.find(artist_id)
  return "no artist with id #{artist_id} found" if artist == nil

  artist_element = Element.new('artist')
  artist_element.add_attribute('id', artist_id)
  name_element = Element.new('name', artist_element)
  name_element.add_text(artist.name)

  artist.cds.each do |cd|
    cd_element = if @deep
      artist_element.add_element(get_cd(cd.id))
    else
      Element.new('cd', artist_element)
    end
    cd_element.add_attribute('id', cd.id)
    cd_element.add_attribute('href', get_resource_url('cd', cd.id)) if not @deep
  end

  artist_element
end
```



32

AJaX

## MusicServer.rb (Cont'd)

```
def get_cd(cd_id)
  cd = Cd.find(cd_id)
  return "no cd with id #{cd_id} found" if cd == nil

  cd_element = Element.new('cd')
  cd_element.add_attribute('id', cd.id)
  cd_element.add_attribute('artistId', cd.artist_id)
  title_element = Element.new('title', cd_element)
  title_element.add_text(cd.title)

  cd.tracks.each do |track|
    track_element = if @deep
      cd_element.add_element(get_track(track.id))
    else
      Element.new('track', cd_element)
    end
    track_element.add_attribute('href',
      get_resource_url('track', track.id)) if not @deep
  end

  cd_element
end
```



33

AJaX

## MusicServer.rb (Cont'd)

```
def get_track(track_id)
  track = Track.find(track_id)
  return "no track with id #{track_id} found" if track == nil

  track_element = Element.new('track')
  track_element.add_attribute('id', track.id)
  track_element.add_attribute('cd_id', track.cd_id)
  track_element.add_attribute('rating', track.rating)
  track_element.add_text(track.name)

  track_element
end

end # class MusicServlet
```



34

AJaX

## MusicServer.rb (Cont'd)

```
# Create WEBrick server.  
# Configure so files in DocumentRoot can be accessed  
# with the URL http://localhost:{SERVLET_PORT}/{file}  
config = {  
  :DocumentRoot => '/AJaX/MusicCollection/web',  
  :FancyIndexing => true, # If URI refers to a directory, list the contents.  
  :Port => SERVLET_PORT  
}  
server = HTTPServer.new(config)  
  
# Add mime type for XHTML.  
mimeTypes = server.config[:MimeTypes]  
mimeTypes['xhtml'] = 'text/html'  
  
# Allow the server to be stopped with Ctrl-c.  
trap('INT') { server.shutdown }  
trap('TERM') { server.shutdown }  
  
server.mount}/#{SERVLET_NAME}", MusicServlet)  
server.start
```



35

AJaX

## 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" />  
  
    <script type="text/javascript" src="sarissa.js"></script>  
    <script type="text/javascript" src="sarissa_ieemu_xpath.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>
```



36

AJaX

## MusicCollection.xhtml (Cont'd)

```
<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>
```



37

AJAX

## MusicCollection.xhtml (Cont'd)

```
<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></option> <!-- XHTML requires at least one option -->
</select>
</td>
</tr>
</table>
```



38

AJAX

## MusicCollection.xhtml (Cont'd)

```
<!-- for debugging -->
<!--p><textarea id="log" rows="20" cols="80"></textarea></p-->

<p><input type="reset" /></p>
</form>
</body>
</html>
```



39

AJAX

## DHTMLUtil.js

```
// This contains utility functions make working with DHTML easier.

// Adds an option to the end of a select.
function addOption(select, option) {
    if (isIE()) {
        select.add(option);
    } else {
        select.add(option, null);
    }
}

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



40

AJAX

## DHTMLUtil.js (Cont'd)

```
// Delete all the rows in a given table except the header row.  
function clearTable(table) {  
    rowCount = table.rows.length;  
    for (i = rowCount - 1; i > 0; i--) {  
        table.deleteRow(i);  
    }  
}  
  
// 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;  
}
```



41

AJAX

## DHTMLUtil.js (Cont'd)

```
// Highlights the characters at the end of an input field  
// starting from a given position.  
function highlightInput(input, start) {  
    totalLength = input.value.length;  
    if (isIE()) {  
        range = input.createTextRange();  
        range.moveStart("character", start);  
        range.select();  
    } else {  
        input.setSelectionRange(start, input.value.length);  
    }  
  
    // Determines if the web browser is IE.  
    function isIE() {  
        var browserName = navigator.appName;  
        return browserName == "Microsoft Internet Explorer";  
    }
```



42

AJAX

## DHTMLUtil.js (Cont'd)

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

// Sends an asynchronous HTTP request to a given URL
// whose response will be sent to a given handler.
function send(url, handler) {
    // XMLHttpRequest is used to send asynchronous HTTP requests.
    // Firefox seems to require creating a new XMLHttpRequest object
    // for each request.
    xhr = new XMLHttpRequest(); // from Sarissa

    xhr.onreadystatechange = handler;
    async = true;
    xhr.open("GET", url, async);
    body = null;
    xhr.send(body);
    return xhr;
}
```

This is the main place where  
**AJAX** appears in this application!  
Don't blink or you'll miss it!



43

AJAX

## MusicCollection.js

```
// 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 = "";

// Holds an XMLHttpRequest object that is used to
// send asynchronous HTTP requests.
var xhr = null;
```



44

AJAX

## 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) {
  // For example, the user may have pressed Ctrl-P to print.
  // At this point ctrlKeyDown could be true and
  // event.keyCode could be the code for 'P'.
  if (!ctrlKeyDown) getArtists(event, component);
  if (event.keyCode == ctrlKeyCode) ctrlKeyDown = false;
}
```



45

AJaX

## MusicCollection.js (Cont'd)

```
// Handles selections of artists in the artist select component.
function artistSelected(component) {
  index = component.selectedIndex;
  value = component.options[index].text;

  // Copy selected value to text input field.
  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
}
```



46

AJaX

## MusicCollection.js (Cont'd)

```
// 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"));
        clearSelect(document.getElementById("cdSelect"));
        clearTable(document.getElementById("trackTable"));
    } else {
        url = baseURL + "artist?starts=" + artistPrefix;
        xhr = send(url, handleArtists);
    }
}
```



OBJECT COMPUTING, INC.

47

AJAX

## MusicCollection.js (Cont'd)

```
// 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";
    xhr = send(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";
    xhr = send(url, handleTracks);
}
```



OBJECT COMPUTING, INC.

48

AJAX

## MusicCollection.js (Cont'd)

```
// 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");
        nodes = doc.selectNodes("/artists/artist"); // from Sarissa

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

        if (nodes.length == 0) return;
    }
}
```



49

AJaX

## 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);  
    addOption(artistSelect, option);
}

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

// Highlight suffix supplied by search.
highlightInput(input, lastArtistPrefix.length);

getCDs();
}
```



50

AJaX

## MusicCollection.js (Cont'd)

```
// 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");
        nodes = doc.selectNodes("/artist/cd"); // from Sarissa

        select = document.getElementById("cdSelect");
        clearSelect(select);
```



51

AJax

## MusicCollection.js (Cont'd)

```
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, false, i == 0);
    addOption(select, option);
}

getTracks(firstId);
}
```



52

AJax

## MusicCollection.js (Cont'd)

```
// 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");
        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);
        }
    }
}
```



53

AJax

## MusicCollection.js (Cont'd)

```
// 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;
}
```



54

AJax

## 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**

- don't send request for artists that match the name typed  
until some amount of time (1 second?) has passed  
without more characters being typed
- 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
  - what caching is supplied automatically by the browser?
- display years after CDs
- add sequence numbers to request and response messages  
so they are paired correctly when there are concurrent requests?



55

AJaX

## Wrap Up (Cont'd)

- Any questions?
- Thank you very much for attending!



56

AJaX