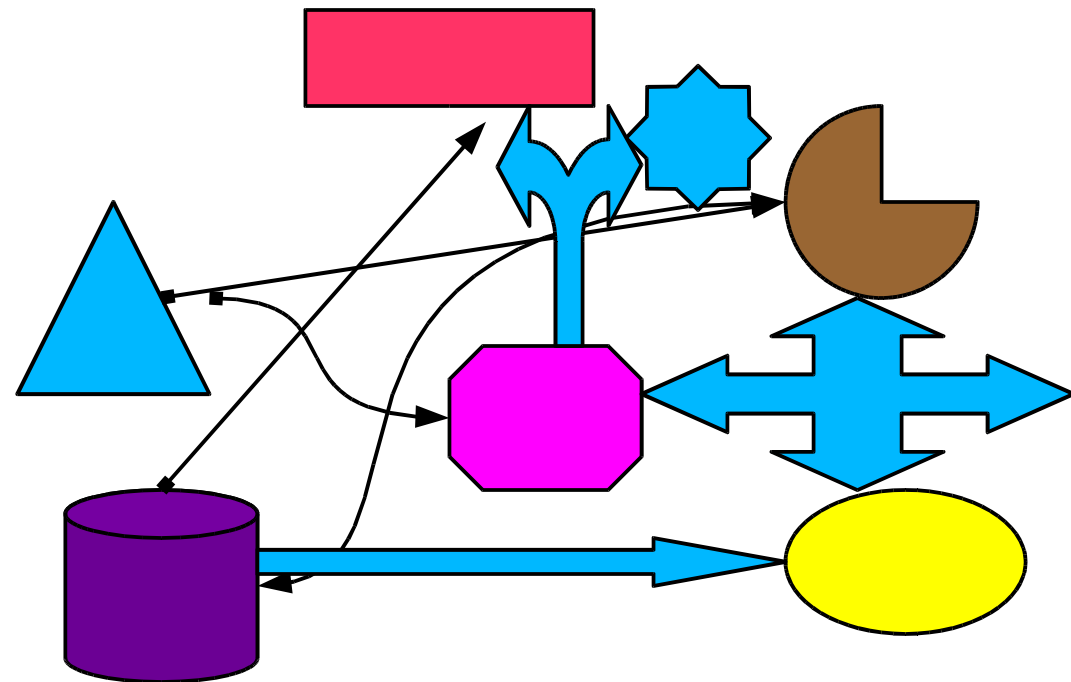


# HTML Forms Simplified

**Alex Kodat**  
**Sirius Software Inc.**



Sirius Software, Inc.

# HTML and HTTP

- The defacto standard for communicating between servers and end-users
- **HTML:** HyperText Markup Language
  - Specifies how data is presented to an end user
- **HTTP:** HyperText Transfer Protocol
  - Specifies how data is exchanged between client (originally browsers) and server
  - Request/response protocol (stateless)
  - Has nothing to do with hypertext



# Presenting Static Data with HTTP/HTML

- HTTP URL sent by browser indicates location of static data
- Server returns static data to browser
  - Often in HTML format for nice text presentation
  - HTML can contain locations of non-text static data (images, audio, video, etc.)
- Static data can return URLs for other static content (links), hence the term *web*
- Really easy
- And there's a lot of relatively static data out there
- This is why the internet took off so quickly



# Complex Retrievals and Updates Using HTTP and HTML

- Server sends HTML <form> with input fields to browser
  - Some of the input fields might be pre-populated with data
- User fills in the fields
  - Javascript might or might not do validation and other fancy stuff as user inputs data
- When user happy with inputs the user *Posts* the form
  - This translates to an HTTP POST request
- Server then responds to the post, at which point all the posted data entered by the user is gone
  - So the server better have saved it somewhere



# The Problems with the HTML/HTTP Update Model

- The format of the posted data is crude
  - application/x-www-form-urlencoded format
  - Originally geared toward complex retrieval applications
  - Hasn't changed in over 15 years!
    - Who says internet technology is quickly evolving?
  - More on this later
- The response to the HTTP Post adds an entry to the browser history
- The posted data is not retained by the browser so the server needs to re-send it if there's a problem



# application/x-www-form-urlencoded Format

- Sets of `<name>=<value>` pairs separated by `&`
  - For example: `Iname=Cureton&fname=Goff`
- Non-alphanumeric characters encoded as hex
  - Except space which is encoded as `+`
  - For example, “This & That” encoded as: `This+%26+that`
- **Does** support unicode
  - For example “sör” (Hungarian for beer) gets sent as: `s%C5%91r`
    - X'C591' is the UTF-8 encoded version of the Hungarian ő



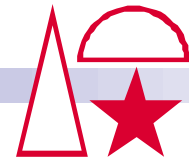
# The Problems With application/x-www-form-urlencoded Format

- No hierarchy of field values
  - So clunky to deal with form field groups
    - And extremely difficult to deal with nested field groups
    - Similar problems to Model 204 fields
- All field values must be sent by browser
  - Even if they're null or unset
    - So bandwidth unfriendly
- Janus Web API for accessing these values sucks
  - Not O-O
  - Doesn't understand unicode
  - Limited to 64K worth of values
    - Names and values stored in real storage not CCATEMP



# The Post Backpage Problem

- Repeated posts for the same URL produce an entry for each post in the browser history
- So, a browser backpage forces user to go through each post
  - Possible, reposting with a warning for each one
- So users shouldn't use backpage
  - This is **lame**
  - Backpage is a nice web feature
- Other solution is to redirect each post to a Get for the same URL
  - Done automatically by `$web_form_done`
  - Not very latency/bandwidth friendly



# There Must Be a Better Way

- Sirius should fix up its \$web API to be O-O and unicode capable and to not have the stupid 64K dependency
- But this still leaves most of the other problems associated with the HTML/HTTP form paradigm
  - Bandwidth/latency unfriendliness
  - Web application coder unfriendliness
  - Backpage unfriendliness



# Is There a Better Way?

- Suppose that a page could have a conversation with the server and send its posted data without losing what's on the screen
  - Like a 3270
  - So no backpage problem
  - So application programmers don't have to reproduce a page to present an error
- Suppose there were HTML tags to nest fieldgroups on a POST
  - So field groups could be dealt with more naturally
  - And nested field groups would be feasible
  - And bandwidth wouldn't be wasted sending null/unset fields
- Sadly, HTML/HTTP don't provide this capability...



# But There IS a Better Way

- You can accomplish the same thing with some (relatively) simple and browser-independent Javascript
- In fact, Sirius has done this
  - Credit to Adrian Jesshope for doing the initial coding
  - Not quite distribution-ready, but, depending on customer reaction, could be made so fairly quickly
  - Already being used in applications being written for OCM BOCES
- Javascript is Model 204 independent but 204's integrated database, programming language, and XML API uniquely position it to take advantage of the Javascript's capabilities.



# So, How Does it Work ? (cont...)

- On any web page you add a `<script>` tag to point to the form processing javascript
  - Truth in advertising: currently, you have to add a handful of `<script>` tags
  - We **will** get it down to one if/when we distribute this
    - And eliminate some chaff
- You also have to add an *onload* attribute to the `<body>` tag
  - This could/should also be eliminated
- When the page is loaded, the Javascript takes over and hooks every form on the page to trap submit (POST) events



## So, How Does it Work ? (...cont)

- When the form is posted it harvests all the form fields and encapsulates inside XML
- It also understands some magic HTML tag attributes for grouping fields
  - Inside container XML elements
- It then posts the XML to the server from the Javascript
  - This is very different from a user post of a page, because it leaves the page in place
- Depending on the response from the server it either goes to a new page or issues some (probably error) messages and stays on the same page



# Example of some XSLT That Uses This Technique

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/*">
    <html>
      <head>
        <title>Form demo</title>
        <script src="webshl_common.js"></script>
        <script src="webshl_box.js"></script>
        <script src="webshl_helper.js"></script>
        <script src="webshl_xml_http_helper.js"></script>
        <script src="webshl_form.js"></script>
      </head>
      <body onload="webshl.common.startup()">
        <center>
          <form method="POST">
            <table>
              <tr>
                <td align="left">Enter some stuff:</td>
                <td align="left"><input name="stuff" type="text"
                  width="20" maxlength="20"/></td>
              </tr>
              ...
            </table>
          </form>
        </center>
      </body>
    </html>
  </template>
</stylesheet>
```



# Extra Work Required On Browser/HTML End

- None



# Web Server Changes With the New Paradigm

- No more `$web_form_parm` calls
- XmlDocument API calls instead
  - XmlDocument loaded with:  
`%requestDoc:loadXml($web_input_content)`
- Rather than rebuilding output page (with possible error messages), web server app builds a response XmlDocument
- And then sends it with: `%postReplyDoc:webSend`
- So XML in and XML out



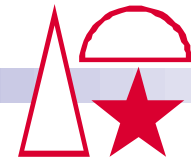
# Grouping Input Fields

- Grouping indicated by a new attribute on any element in the form
  - Element name is *sirfgElement*
  - Value indicates the name of the container element for the input fields on a POST
- Usually the *sirfgElement* attribute will be on a `<tr>` or `<div>` element



# Example of some XSLT That Groups Input Fields

```
<tr sirfgElement="subcategory">
  <td align="left">
    <xsl:if test="$id != ''">
      <input name="id" type="hidden" value="{ $id }"/>
    </xsl:if>
    <input name="delete" type="checkbox" value="Y"/>
  </td>
  <td align="left">
    <input name="description" type="text" size="40" maxlength="100"
      value="{ $description }"/>
  </td>
  <td align="right">
    <input name="weight" type="text" size="6" maxlength="6" value="{ $weight }"/>
  </td>
  <td align="right">
    <input name="maxScore" type="text" size="6" maxlength="6" value="{ $maxScore }"/>
  </td>
  <td align="right">
    <div style="font-weight: lighter;" name="percent">
      0%
    </div>
  </td>
</tr>
```



# Sample Post Data With Nested Form Fields

```
<postRequest>
  <id>12</id>
  <description>Final exam</description>
  <maxScore>100</maxScore>
  <subcategory>
    <id>13</id>
    <description>Multiple guess questions</description>
    <weight>30</weight>
    <maxScore>100</maxScore>
  </subcategory>
  <subcategory>
    <id>14</id>
    <description>Penmanship</description>
    <weight>10</weight>
    <maxScore>100</maxScore>
  </subcategory>
  <subcategory>
    <id>15</id>
    <description>Essay question</description>
    <weight>60</weight>
    <maxScore>100</maxScore>
  </subcategory>
</subcategory/>
</subcategory/>
</postRequest>
```



# Processing Post Data With Nested Form Fields

```
%subcategoryNodes = %postRequest:selectNodes('subcategory')
for %i from 1 to %subcategoryNodes:count
    %subcategoryNode = %subcategoryNodes(%i)
    %formId           = %subcategoryNode:valueDefault('id')
    %formDescription = %subcategoryNode:valueDefault('description'):unspace
    %formWeight      = %subcategoryNode:valueDefault('weight'):unspace
    %formMaxScore    = %subcategoryNode:valueDefault('maxScore'):unspace
    %formDelete      = %subcategoryNode:valueDefault('delete'):unspace
    ...
end for
```



# Advantages of Grouping Input Fields

- Server code much tidier and more sensible
  - Loop is over group not one group member
- Access more efficient
  - Elements of the the group accessed relative to group container element
- Nested groups possible
- Null values don't need to be sent
  - Empty text boxes, unchecked checkboxes, etc.
  - Because don't need placeholders
  - So bandwidth friendly
  - But maybe should be a Javascript option?



# Server Response to Javascript PostRequest

- An XML document with <postResponse> outer element
- Subelements can tell browser the next action
  - Navigate (redirect) to a new page
  - Display errors and let users correct errors
  - Put up a “Do You Really Want To”



# Post Reply To Navigate to a New Page

- `<navig>` element value indicates the URL to which the browser should redirect
  - Has lowest precedence among `<postReply>` subelements
  - So, only honored if nothing else in post reply
- Probably should be the normal response to a successful update operation
  - Redirect should be to a page that verifies the result of the update
  - Must be used instead of `redirect ($web_redirect)` because Javascript is getting the response, not the browser



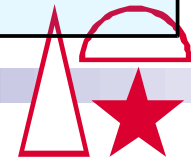
# Post Reply to Redirect to New Page: User Language and Output XML

```
...
%postReply          is object xmlDoc
%postReplyNode      is object xmlNode
...
%postReply = new
%postReplyNode = %postReply:addElement('postReply')
...
commit

%postReplyNode:addElement('navig', 'gradebook_layout.xml?' with -
                           %gradebookSection:urlParameters)

%postReply:webSend
stop
...
```

```
<postReply>
  <navig>
    gradebook_layout.xml?schoolYear=2010&districtNo=600018-
    &schoolNo=0012&courseCode=0181&sectionNo=004&markPeriod=1
  </navig>
</postReply>
```



# Post Reply To Indicate Errors

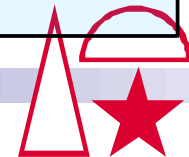
- One or more `<error>` elements with *msg* attributes
  - Can contain one or more `<tag>` subelements that indicate the form fields to be tagged with the error
  - Focus or mouse-over for the tagged fields will show the error
  - Input area flagged with red for tagged fields
- User Language doesn't need to repopulate the form since all the data is still in the browser



# Post Reply to Indicate Errors: User Language and Output XML

```
...
%postReply = new
%postReplyNode = %postReply:addElement('postReply')
...
%formMaxScore      = %subcategoryNode:valueDefault('maxScore'):unspace
...
*****
* AddError is a local method that adds elements and increments      *
* error count.                                                       *
*****
if %formMaxScore eq '' then
    %postReplyNode:addError('Max score required', 'maxScore', %i + 1)
elseif %gradebookCategory:maxScore le 0 then
    %postReplyNode:addError('Max score must be > 0', 'maxScore', %i + 1)
end if
...
if %errors then
    %postReply:webSend('INDENT 2')
    stop
end if
```

```
<postReply>
  <error msg="Max score required">
    <tag field="maxScore" occ="5"/>
  </error>
</postReply>
```




# Post Reply to Indicate Errors: Browser View (Needs some Work)

WEST GENESEE CENTRAL SCHOOLS Welcome Screen

Update gradebook category Quizzes for A.P. US HIST, section 004 in marking period 1 BOCES-CNYRIC

The page at <https://sis.sirius-software.com> says:

 There were errors on the page. No updates have been saved. Correct the errors and re-submit the request or leave the page and discard your changes. The errors on this page were:

Max score required

			100	Score Percent
<input type="checkbox"/>	Quiz 3	40	100	30.77%
<input type="checkbox"/>	Quiz 4	10	100	7.69%
<input type="checkbox"/>	Quiz 1	50	100	38.46%
<input type="checkbox"/>	Quiz 7	30		23.08%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%



# Post Reply To Send a “Do You Really Want To”

- One or more `<dyrwt>` elements
  - Can contain one or more `<button>` subelements that indicate the buttons to appear in the DYRWT window
    - ➔ Usually will just be “Yes” or “No”
  - UL code should check if user has responded that she “really wants to” before sending a DYRWT
- User Language doesn't need to repopulate the form since all the data is still in the browser
- Typical uses:
  - Record locking issues
  - Data updated under user
  - User performing “questionable” operation



# Post Reply “Do You Really Want To”: User Language and Output XML

```
...  
%formMaxScore      = %subcategoryNode:valueDefault('maxScore'):unspace  
...  
if %orphanCategories:count then  
  %dyrwtOrphan = %postRequest:valueDefault('orphanCategories'):left(2,pad='')  
  if %dyrwtOrphan eq 'N' then  
    %postReply:webSend  
    stop  
  end if  
  if %dyrwtOrphan ne 'Y' then  
    %postReplyNode:addDyrwt('orphanCategories', -  
                            'Do you really want to delete categories ' with -  
                            'for which students have grades?')  
  end if  
end if  
...  
if %errors then  
  %postReply:webSend('INDENT 2')  
  stop  
end if
```

```
<postReply>  
  <dyrwt name="orphanCategories" msg="Do you really want to delete categories for-  
    which students have grades?">  
    <button value="Y">Yes</button>  
    <button value="N">No</button>  
  </dyrwt>
```



# Post Reply to Indicate DYRWT: Browser View (Needs some Work)

The screenshot shows a Mozilla browser window displaying a web application for 'WEST GENESEE CENTRAL SCHOOLS'. The page title is 'Update gradebook category Quizzes for A.P. US HIST, section 004 in marking period 1'. A confirmation dialog box titled 'Do You Really Want To - Mozilla' is open, asking 'Do you really want to delete categories for which students have grades?' with 'Yes' and 'No' buttons. The background page shows a form with the following details:

- District: WEST
- School: WEST
- Course: A.P. US
- Category: Quizzes
- Max score: 100

Below the form is a table with columns: 'Delete Subcategory descri', 'Weight', 'Max', 'Score', and 'Percent'. The table contains the following data:

Delete Subcategory descri	Weight	Max	Score	Percent
<input checked="" type="checkbox"/> Quiz 3		100		30.77%
<input type="checkbox"/> Quiz 4	10	100		7.69%
<input type="checkbox"/> Quiz 1	50	100		38.46%
<input type="checkbox"/> Quiz 7	30	100		23.08%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%
<input type="checkbox"/>				0%

The browser's status bar at the bottom shows 'Done' and 'sis.sirius-software.com'.

# Advantages of New Posting Technology

- Eliminates need to re-build posted page on errors
- Fixes backpage issues associated with posts
- Fixes double post problem
- Facilitates form field group processing
  - Reduces quantity of data transferred
- Works around Sirius web API shortcomings
- Simple UL interface
  - Just produce some XML
- Simple HTML/XSLT interface
  - Just a few `<script>` tags and an *onload* attribute in your document
    - Ultimately just one `<script>` tag



# Issues With New Posting Technology

- Javascript packaging needs some work
- Pop-up windows for errors and DYRWTs still need a bit of work
- Maybe want a class to simplify generation of Post Reply
  - But it's so easy without one, it's not clear if it's worth the trouble
- Cough, cough, documentation.. cough, cough
- Should be packaged up in SIRIUS file by, say, July
  - Available upon request earlier
- Needs a catchy name?

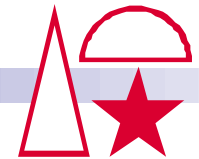


# Conclusions

- New posting technology makes writing updating web apps much easier
- We are eating our own dogfood
  - Using the new posting technology for code we're developing for OCM BOCES



# Questions? Comments?



Sirius Software, Inc.