ArcGIS Server Java

ICEfaces - Esri Integration Guide

Why should I combine the ESRI WebADF with ICEfaces technology?

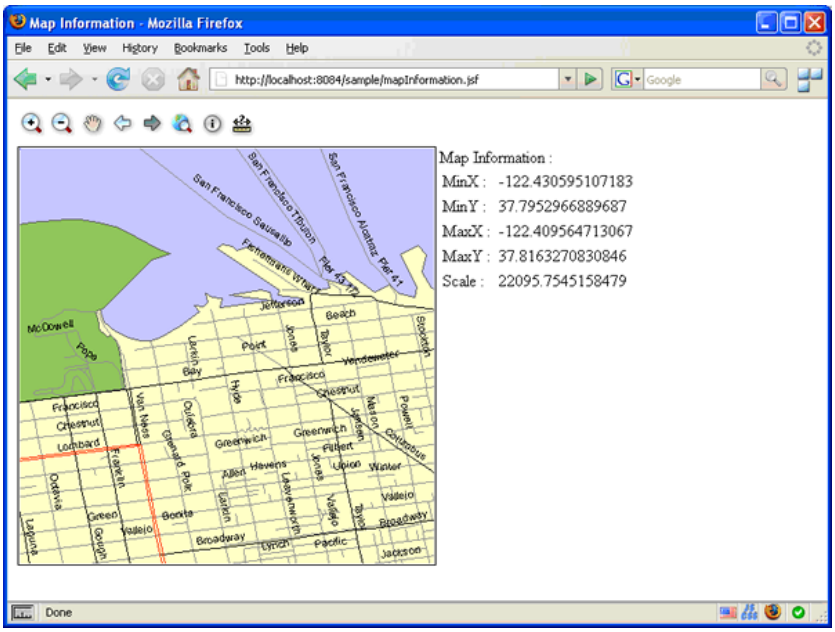
- Only the modified parts will be updated out of the box. Fully AJAX enabled.
  - You zoom into your map and only the modified map image would be reloaded. Not the entire webpage is reloaded as would be for common web applications.
  - Smooth, incremental page updates that do not require a full page refresh to achieve presentation changes in the application. Only elements of the presentation that have changed are updated during the render phase (D2D).
  - User context preservation during page update, including scroll position and input focus. Presentation updates do not interfere with the user's ongoing interaction with the application.
- You don't have to handle the AJAX communication
  - No need to create your data into an XML representation on server-side
  - No need to parse the XML data on client-side
  - No need to handle the synchronous XML communications
- Server-initiated asynchronous presentation update
  - Initiate a client rendering from the server side. E.g. after editing the geometry of a feature, initiate an asynchronous map refresh on all clients (Intranet + Internet).
  - Standard JSF applications can only deliver presentation changes in response to a user-initiated event, typically some type of form submit. ICEfaces introduces a trigger mechanism that allows the server-resident application logic to push presentation changes to the client browser in response to changes in the application's state. This enables application developers to design systems that deliver data to the user in a near-real-time asynchronous fashion.
- Work directly with the W3C Html elements
  - ICEfaces Direct-to-DOM (D2D) rendering is just what it sounds like—the ability to render a JSF component tree directly into a W3C standard DOM data structure. ICEfaces provides a Direct-to-DOM Render Kit for the standard HTML basic components available in JSF.
  - One of the key features of Direct-to-DOM rendering is the ability to perform incremental changes to the DOM that translate into in-place editing of the page and result in smooth, flicker-free page updates without the need for a full page refresh.

What's the code difference of ESRI WebADF with/without ICEfaces technology?

Take a look of the complexity in the appendixes of this documentation or on the Internet for the following two samples provided by Esri.

- MapInformation
  - http://enddoc.esri.com/arcobjects/9.2/Java/server_samples/map-information/example.html

- MapZoom
  - http://enddoc.esri.com/arcobjects/9.2/Java/server_samples/map-zoom/example.html



With the ICEfaces – Esri integration code, you don't have to deal with code like:

- JavaScript
  - var xmlHttp = EsriUtils.sendAjaxRequest(url, params, true, function(){
- Java
  - public PhaseId getPhaseId(){ …. }
- XML
  - Document doc = XMLUtil.newDocument();
    Element responseElement = XMLUtil.createElement(doc, "response", null, null);

On the 2 following pages, you will discover the needed code to implement the same functionalities/features as in the Esri samples.
Compare it directly with the appendixes.

Dive into the ICEfaces - ESRI code

Map INFORMATION example

http://edndoc.esri.com/arcobjects/9.2/Java/server_samples/map-information/example.html

As you will see, it's very easy to get working the map information sample. No JavaScript, no xml-transfer. The code on this page is replacing the whole sample on the link above.

HTML code:

```
<pch:agscontext webContext="#{esriWebContext}"/>
<pch:agstool displayType="IMAGE_AND_TEXT" mapId="agsmap1" toolText="zoomIn"
serverMethod="#{pchMapBean.zoomIn}" clientAction="EsriMapRectangle" />
<pch:agstool displayType="IMAGE_AND_TEXT" mapId="agsmap1" toolText="zoomOut"
serverMethod="#{pchMapBean.zoomOut}" clientAction="EsriMapRectangle"/>
 <ice:outputText value="MinX: #{mapInformation.minX}"/>
<ice:outputText value="MinY: #{mapInformation.minY}"/>
<ice:outputText value="MaxX: #{mapInformation.maxX}"/>
<ice:outputText value="MaxY: #{mapInformation.maxY}"/>
<ice:outputText value="MapScale: #{mapInformation.mapScale}"/>
<pch:agsmap id="agsmap1" webMap="#{esriWebContext.webMap}"/>
```

JAVA code:

```
public class MapInformation implements WebContextObserver, WebContextInitialize {
   NumberFormat decFormatter = NumberFormat.getNumberInstance(new Locale("fr",
"LU"));
   private double minX, minY, maxX, maxY;
   private double mapScale;
   public String getMinX() {
      return decFormatter.format(minX);    }
   public String getMinY() {
      return decFormatter.format(minY);    }
   public String getMaxX() {
      return decFormatter.format(maxX);    }
   public String getMaxY() {
      return decFormatter.format(maxY);    }
   public String getMapScale() {
      return decFormatter.format(mapScale);    }
   public void init(WebContext webContext) {
      webContext.addObserver(this);
      calculateMapInfo(webContext);
   }
   public void destroy() {    }
   public void update(WebContext webContext, Object o) {
      calculateMapInfo(webContext);
   }
   private void calculateMapInfo(WebContext webContext) {
      WebExtent currentExtent = webContext.getWebMap().getCurrentExtent();
      this.minX = currentExtent.getMinX();
      this.minY = currentExtent.getMinY();
      this.maxX = currentExtent.getMaxX();
      this.maxY = currentExtent.getMaxY();
      this.mapScale = webContext.getWebMap().getMapScale();
   }
}
```

Faces-context.xml:

```
<managed-bean>
   <managed-bean-name>mapInformation</managed-bean-name>
   <managed-bean-class>lu.etat.pch.gis.test.MapInformation</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Map ZOOM example

http://edndoc.esri.com/arcobjects/9.2/Java/server_samples/map-zoom/example.html

As you will see, it's very easy to get working the map zoom sample. No JavaScript, no xml-transfer. The code on this page is replacing the whole sample on the link above resp. the code in the appendix.

HTML code:

```
<pch:agscontext webContext="#{esriWebContext}"/>
<ice:selectOneMenu partialSubmit="true"
valueChangeListener="#{mapZoom.mapZoomComboChanged}">
        <f:selectItem itemValue="0.2"/>     <f:selectItem itemValue="0.4"/>
        <f:selectItem itemValue="0.6"/>     <f:selectItem itemValue="0.8"/>
        <f:selectItem itemValue="1.0"/>     <f:selectItem itemValue="1.2"/>
        <f:selectItem itemValue="1.4"/>     <f:selectItem itemValue="1.6"/>
        <f:selectItem itemValue="1.8"/> o   <f:selectItem itemValue="2.0"/>
</ice:selectOneMenu>
<ice:panelGrid columns="2">
        <ice:outputText value="MinX: #{mapInformation.minX}"/>
        <ice:outputText value="MinY: #{mapInformation.minY}"/>
        <ice:outputText value="MaxX: #{mapInformation.maxX}"/>
        <ice:outputText value="MaxY: #{mapInformation.maxY}"/>
        <ice:outputText value="MapScale: #{mapInformation.mapScale}"/>
</ice:panelGrid>
<pch:agsmap id="agsmap1" webMap="#{esriWebContext.webMap}"/>
```

JAVA code:

```
public class MapZoom {
   private WebContext webContext;
   public void setWebContext(WebContext webContext) {
      this.webContext = webContext;
   }
   public void mapZoomComboChanged(ValueChangeEvent vcE) {
      String zoomFactorValue = (String) vcE.getNewValue();
      if (webContext != null && webContext.getWebMap() != null) {
         WebMap webMap = webContext.getWebMap();
         WebExtent webExtent = webMap.getCurrentExtent();
         double zoomFactor = Double.parseDouble(zoomFactorValue);
         webExtent.expand(zoomFactor);
         webMap.setCurrentExtent(webExtent);
         webContext.refresh();
      }
   }
}
```

Faces-context.xml:

```
<managed-bean>
   <managed-bean-name>mapZoom</managed-bean-name>
   <managed-bean-class>lu.etat.pch.gis.test.MapZoom</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>webContext</property-name>
      <value>#{esriWebContext}</value>
   </managed-property>
</managed-bean>
```

!! NOTHING MORE TO CODE OR TO HANDLE !!

Roadmap to get working my ICEfaces Esri integration:

Mixing up the ICEfaces technology with the existing ESRI components doesn't work out of the box.

The ESRI components are fully built up to work with the standard basic AJAX communication framework. The XML data is encoded on server-side by Java-code, transferred to the client where it will be parsed and applied on the html elements by JavaScript.

See the examples:

- http://edndoc.esri.com/arcobjects/9.2/Java/server_samples/map-information/example.html
- http://edndoc.esri.com/arcobjects/9.2/Java/server_samples/map-information-2/example.html
-

With ICEfaces, you don't have this AJAX communication. On the server-side, you modify your desired client user-interface directly with Java-Code.

This was demonstrated in the previous example of MapInformation and MapZoom.
As you have seen, there is no XML creation/parsing or JavaScript code.

For using the ICEfaces advantages, I had to rewrite the Esri components. At the backend, the complete WebADF API is used. The compatibly with Esri WebADF is guaranteed.

Actually I have rewritten these components with his basic functionalities:

- <a:context>
- <a:map>
- <a:toc>
- <a:task>
- <a:command>
- <a:tool>

I used my own namespace and components names.
So the resulting ICEfaces-compatible JSF components are:

- <pch:agscontext>
- <pch:agsmap>
- <pch:agstoc>
- <pch:agstask>
- <pch:agscommand>
- <pch:agstool>

What is the actual re-development status of the different components?

I have actually implemented the basic functionalities and attributes to demonstrate a POC (prove of concept) and to build up a first demonstration application.

Below you find a detailed listing of the missing functionalities and attributes compared to the standard ESRI WebADF library.

- **<pch:agscontext>** (replacement for <a:context>)
  Non-implemented attributes:
  - preserve

- **<pch:agsmap>** (replacement for <a:map>)
  Non-implemented functionalities:
  - This component doesn't yet handle tiles and map caches. Actually it can only show up non-cached map-services.

  Non-implemented attributes:
  - scaleBar
  - width
  - height
  - xslFile
  - style / styleClass

- **<pch:agstoc>** (replacement for <a:toc>)
  The TOC is rewritten on some special way. Its whole representation is defined in a separate faclet-file. There is no new renderer created. See :/WEB-INF/taglib/pchToc.xhtml

- **<pch:agstask>** (replacement for <a:task>)
  The task was a more complex component to rewrite. I don't have implemented task framework yet. On the other side, I have added some basic helpers to work better with the Task framework
  Non-implemented functionalities:
  - Only the default and the table layout are implemented. The "AbsoluteLayout" isn't yet implemented

  Non-implemented attributes:
  - taskInfo (uses getTaskInfo() on task class
  - xslFile
  - style / styleClass
  - windowingSupport
  - clientPostBack

- **<pch:agscommand>** (replacement for <a:command>(
  Non-implemented functionalities:
  - You can't yet define JavaScript code as 'action' or 'onClick' which should directly run on client-side

  Non-implemented attributes:
  - onClick
  - style / styleClass
  - clientPostBack (not needed)
  - toolTip
  - showLoadingImage

- **<pch:agstool>** (replacement for <a:tool>(
  Non-implemented functionalities:
  - You can't yet define JavaScript code as 'action' or 'onClick' which should directly run on client-side

  Newly implemented attributes:
  - mapId
    You don't have to put a tool in a toolbar which contains the relation to the map. Place your tools around the whole page and link the tool directly to the map.

  Non-implemented attributes:
  - serverMethid
  - style / styleClass
  - clientPostBack (not needed)
  - toolTip
  - showLoadingImage
  - cursor
  - lineColor / lineWidth

How should I start with the development of an Icefaces-ESRI application?

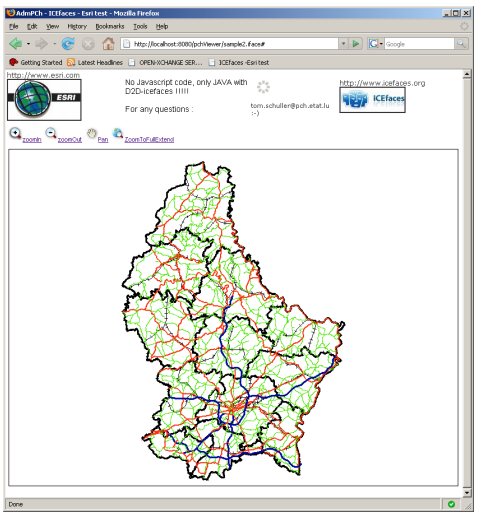
To getting started very quickly with my library, I have provided some step-by-step samples. You can also find all the following samples in the WAR-file.

- Sample1.xhtml: **SimpleMap**
  - (http://www2.pch.etat.lu/pchViewer/sample1.iface)
    As in a simple Esri WebADF web application, you will have to place a context control and a map control on your page.
    The context control initiates your "WebContext" during your session.
    The WebContext is configured in your faces-context.xml file.
    The map control is displaying up your map.

```
<pch:agscontext
        webContext="#{esriWebContext}"/>
<pch:agsmap
        webMap="#{esriWebContext.webMap}"/>
```

- Sample2.xhtml: **SimpleMap + Tools + Commands**
  - (http://www2.pch.etat.lu/pchViewer/sample2.iface)
    You can simple add a map tool or command. The map tool/command is linked directly to map with the mapId attribute. This isn't provided by the Esri library, but eliminates the need of the Esri toolbar.

```
<pch:agstool
        displayType="IMAGE_AND_TEXT"
        mapId="agsmap1" toolText="zoomIn"
        serverMethod="#{pchMapBean.zoomIn}"
        clientAction="EsriMapRectangle"
        defaultImage="./images/tasks/maptools/zoomin.png"
        hoverImage="./images/tasks/maptools/zoominU.png"
        selectedImage="./images/tasks/maptools/zoominD.png"
        disabledImage="./images/tasks/maptools/zoominX.png"
        />
<pch:agscontext webContext="#{esriWebContext}"/>
<pch:agsmap webMap="#{esriWebContext.webMap}"/>
```



- Sample3.xhtml: **SimpleMap + TableOfContentes**
  - (http://www2.pch.etat.lu/pchViewer/sample3.iface)

    package lu.etat.pch.gis.arcgisserver.toc;

    My TOC doesn't rely on the WebToc. It also doesn't use the TOCFunctionality. To retrieve the layer names, it's looping through all AGSMapFunctionalities. This should be changed in a next version to work directly with the WebToc and the TOCFunctionality.
    The UI of the TOC is defined in /WEB-INF/taglib/pchTOC.xhtml. This makes it very easy to define additional UI components like a menu-context for each layer.

```
<pch:pchToc
    treeModel="#{pchTocTreeBean.model}"
    webMap="#{esriWebContext.webMap}"/>
```

- Sample4.xhtml: **SimpleMap + GeoBookMark**
  - (http://www2.pch.etat.lu/pchViewer/sample4.iface)

    package lu.etat.pch.gis.arcgisserver.bookmarks;

    The GeoBookmark is a small utility which reads from a text file (comma separated) some locations points and renders it in an auto-complete ComboxBox. Once a geobookmark is selected, you can zoom or pan to it.

```
<pch:pchGeoBookmark
    bookmarkBean="#{geoBookmarkBean}"/>
```

- Sample5.xhtml: **SimpleMap + UserLogin (Security)**
  - (http://www2.pch.etat.lu/pchViewer/sample5.iface)

    package lu.etat.pch.gis.arcgisserver.security;

    I also implemented a very basic login logic. After a login or logout, the WebContext is reinitialized. During initialization of the WebContext, the PchSecurityCheckFunctionality is checking all the mapFunctionality-layers with the authorized layerslist of the actually logged user. The non-authorized map layers will be removed from the functionalities.
    Actually the security is only checking the layer name, in future it should also work by authorized view regions. E.g. the map service "map1" can be showed for UserA for the whole region, the UserB should only see an extract of it over a region defined by a Polygon.
    Try the demos by logging you in as "esri" or "admin".

```
<pch:pchLogin
    loginBean="#{pchLoginBean}"/>
```

- Sample6.xhtml: **SimpleMap + Tasks**
  - (http://www2.pch.etat.lu/pchViewer/sample6.iface)
    package com.icesoft.applications.faces.auctionMonitor;
    The sample6 illustrated the usage of the Task Framework provided by Esri.

```
<pch:agstask
    mapId="agsmap1"
    value="#{measureTask}"/>
```

- Sample7.xhtml: **SimpleMap + Chat**
  - (http://www2.pch.etat.lu/pchViewer/sample7.iface)
    package com.icesoft.applications.faces.auctionMonitor;
    The sample6 illustrated the capabilities of the server initiated rendering capabilities of ICEfaces. Simply load the sample from 2 or more different computers, login to the chat and start chatting. As one user is entering new message, all the other users will be notified.

- Sample8.xhtml: **SimpleMap + ArcSDE edit Task**
  - (http://www2.pch.etat.lu/pchViewer/sample8.iface)
    package com.icesoft.applications.faces.auctionMonitor;
    Prior using the ArcSDE editing task, you will have to configure your ArcSDE connection pool in 'faces-context.xml' and for using the pre-build ArcSDE editing "AccidentTask" you will have to create an SDE point layer with the following fields in the configured ArcSDE connection.
    You will have to log in as "admin" user to get access to the layer.

Field Name	Data Type
OBJECTID	Object ID
local_route	Text
local_pk	Double
nature	Text
part_type	Text
part_nbr	Short Integer
cause	Text
suite	Text
date	Date
local_x	Double
local_y	Double
remarque	Text
SHAPE	Geometry

The actual implemented GISo functionalities on the demo application:

See the file demo.xhtml (http://www2.pch.etat.lu/pchViewer/demo.iface)

- Draw text on map
  - User enters text to display
  - User defines location to display
- Measure
  - Distance by Polyline
  - Area/Perimeter by Polygon
- Resize current display map
- Send current map config by email
  - Includes current web extent
  - Includes Visible layers
- GeoBookmarks
  - Loads bookmarks from text file
  - Auto-completion combo box
  - Zoom to selection
  - Pan to selection
- Table of contents TOC
  - Show/hide layers
  - Context menu (right mouse click)
    - Select layer
    - Zoom To Layer Extent
    - Attribute Table
      - View Attributes for current extent
      - Export Attributes for current extent as Ms Excel file
    - Export Data
      - As AutoCAD file
        - Many thanks to "rammi@caff.de" with his library http://www.caff.de/dxfviewer
- User login
  - Define layers to show up by user.
- Chat functionality
  - Get notified on data modifications and automatically get a map refresh
  - Chat with other online users
- Direct ArcSDE Edit Framework (check out the sources)
- ---
- ….

Conclusion

I hope that was able to give you an idea of the simplicity of development with ICEfaces compared to the traditional way of Esri (see appendix).
Any feedbacks or ideas are welcome.

All the sources are included in the war-file. I hope that we can build up a powerful Java community in Palm Springs and discuss about improvements and ideas to build up powerful Esri Java web applications.

You can find any further developments or progresses about this project at:
http://www2.pch.etat.lu/pchViewer

Give your ESRI application the right face, an "ICEface"

For any question or remarks:
tom.schuller@pch.etat.lu
http://www2.pch.etat.lu/pchViewer

Source:

- ICEfaces Developer Guide (http://www.icefaces.org)
- Esri EDN documentation (http://edn.esri.com)

Appendix : (source ArcGIS Server93beta documentation)

Excerpt of the new ArcGIS9.3beta-Server documentation concerning AJAX integration and support in WebADF.

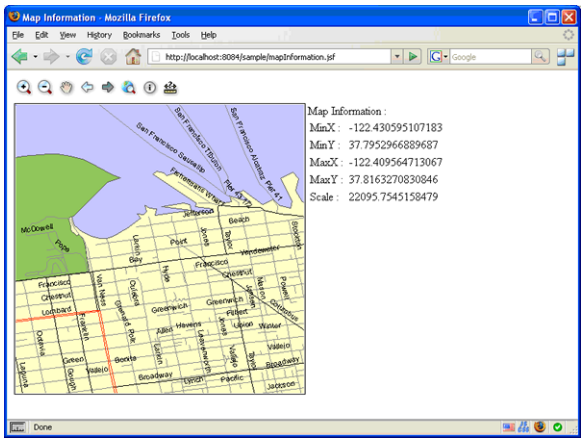
AJAX integration and support

This document was published with ArcGIS 9.3.
Summary: The term “AJAX” was introduced by Jesse James Garrett. It stands for Asynchronous JavaScript and XML and describes a Web programming model that allows content on a browser page to be updated without full postback to the Web server and the re-rendering of elements on the page.

Use case 1: Map Information application

This use case shows how to accomplish the following tasks:

- Send a request to the server using XMLHttpRequest
- Use a PhaseListener to process the request and render an XML response
- Parse the XML response to update elements on the browser page using JavaScript



The Map Information application allows you to update content on the browser page based on the change in the current extent of the map. When running this application and zooming in and out or panning the map, a request is sent to the server and the extent and scale of the map is retrieved and displayed.

mapInformation.jsp

The mapInformation.jsp page contains a map control and a table with several placeholders to display the extent and scale of the currently visible map as shown in the following code sample.
The call to the init() JavaScript function initializes some behavior in the application.

[XML]
```
<body onload="init()">
  <a:map id="map" value="#{mapContext.webMap}" … />
  Map Information :
  <table><tbody>
    <tr>
      <td>MinX : </td><td><span id="minx"></span></td>
    </tr><tr>
      <td>MinY : </td><td><span id="miny"></span></td>
    </tr><tr>
      <td>MaxX : </td><td><span id="maxx"></span></td>
    </tr><tr>
      <td>MaxY : </td><td><span id="maxy"></span></td>
    </tr><tr>
      <td>Scale : </td><td><span id="map-scale"></span></td>
    </tr>
  </tbody></table>
</body>
```

mapInformation.js

The mapInformation.js file defines a set of JavaScript functions used in this application. The init() function sets up a listener to the map control to be notified whenever the map is updated as shown in the following code sample.

[Java]
```
map.addUpdateListener("mapInformationListener",
updateMapInformationRequest);
```
The updateMapInformationRequest() function allows you to send requests to the server to obtain the required map information. Before sending the request to the server, however, you need to set up parameters for the request. First, you need to get the uniform resource locator (URL) of the server to which you are sending the request as shown in the following code sample.
[Java]
```
var url = EsriUtils.getServerUrl(map.formId);
```
Build the request parameters that are to be sent to the server for processing the request. As part of the list of parameters, include the parameters from form elements in which the map is rendered as shown in the following code sample.
[Java]
```
var params = "mapinformation=mapinformation&formId=" + map.formId +
"&mapId=" +
    map.id + "&" + EsriUtils.buildRequestParams(map.formId);
```
Send the request to the server. In addition to passing the server URL, the parameters string, and either the POST or GET function, pass the callback function, which must be called when the XMLHttpRequest object is updated as shown in the following code sample.

[Java]
```
var xmlHttp = EsriUtils.sendAjaxRequest(url, params, true,
function(){
    updateMapInformationResponse(xmlHttp);
}

);
```
The parameters that are sent to the server for processing are as follows:

- Mapinformation—Helps the appropriate phase listener identify whether it's responsible for processing the request
- map.formId—Helps the phase listener identify the form in which the map control is rendered
- map.id—Helps the phase listener obtain the map control from the form

MapInformationPhaseListener.java

MapInformationPhaseListener is the class that processes the request and gets the map control and its current extent and scale information. It then renders this information as XML and returns a response to the client.

The MapInformationPhaseListener class implements the methods defined in the PhaseListener interface. PhaseListener.getPhaseId() tells the JSF implementation about the phase during which this PhaseListener is to be notified as shown in the following code sample.

[Java]
```
public PhaseId getPhaseId(){
    return PhaseId.APPLY_REQUEST_VALUES;
}
```
The Apply Request Values phase is the phase where every control retrieves updated values from the request parameter.

The PhaseListener.beforePhase() method is not used because the values of the controls must be updated before performing any updates as shown in the following code sample.

[Java]
```
public void beforePhase(PhaseEvent phaseEvent){
    //Do nothing.
}
```
The PhaseListener.afterPhase() method processes the request as shown in the following code sample.

[Java]
```
public void afterPhase(PhaseEvent phaseEvent){
    // Determine if this PhaseListener is responsible for
    // processing the request.
    // Process the request.
    // Render and return a response.
}
```
Determine if the request is to be processed by this PhaseListener. This can be done by checking if a specific parameter is present in the request. For example, with MapInformationPhaseListener, check whether the request parameter mapinformation is equivalent to “mapinformation” as shown in the following code sample.

[Java]
```
FacesContext facesContext = phaseEvent.getFacesContext();
ExternalContext externalContext = facesContext.getExternalContext();
Map params = externalContext.getRequestParameterMap();
if (!"mapinformation" .equals(params.get("mapinformation")))
    return ;
```
The facesContext object refers to the FacesContext instance, of which this PhaseListener is a part. Thus, it has access to all the components that are rendered in the view root of the page. Use the request parameters to find the components of interest for your processing. For example, you can find the form and map control on the page as shown in the following code sample.
[Java]
```
String formId = (String)params.get("formId");
UIComponent form = facesContext.getViewRoot().findComponent(formId);
String mapId = (String)params.get("mapId");
MapControl mapControl = (MapControl)form.findComponent(mapId);
WebMap map = mapControl.getWebMap();
```
Perform the necessary processing with the desired component and GIS business object as shown in the following code sample.
[Java]
```
WebExtent extent = map.getCurrentExtent();
double scale = map.getMapScale();
```
As part of the AJAX response, an XML response must be sent back to the client. Create the XML document and populate it as shown in the following code sample.

[XML]
```
Document doc = XMLUtil.newDocument();
Element responseElement = XMLUtil.createElement(doc, "response",
null, null);
XMLUtil.createElement("minx", Double.toString(extent.getMinX()),
responseElement);
XMLUtil.createElement("miny", Double.toString(extent.getMinY()),
responseElement);
XMLUtil.createElement("maxx", Double.toString(extent.getMaxX()),
responseElement);
XMLUtil.createElement("maxy", Double.toString(extent.getMaxY()),
responseElement);
XMLUtil.createElement("map-scale", Double.toString(scale),
responseElement);
```
Once the XML response document has been created and populated with the required information, write it back to the client using AJAXUtil.writeResponse() as shown in the following code sample.
[Java]
```
AJAXUtil.writeResponse(facesContext, doc);
```

When the processing of the request and the response is complete, notify the JSF implementation that the rendering of the response is complete so that the JSF lifecycle terminates at the end of this phase as shown in the following code sample.

[Java]
```
facesContext.responseComplete();
```
The MapInformationPhaseListener class has completed its work. To register a phase listener in the JSF lifecycle, it must be included in faces-config.xml as shown in the following code sample.

[Java]
```
 < lifecycle >  < phase - listener >

com.esri.arcgis.sample.mapinformation.MapInformationPhaseListener <
/ phase -
     listener >  <  / lifecycle >
```
The response is returned to the updateMapInformationResponse JavaScript function. This function is now responsible for parsing the XML and updating the appropriate HTML elements on the page as shown in the following code sample.

[XML]
```
var xml = xmlHttp.responseXML;
document.getElementById("minx").innerHTML =
xml.getElementsByTagName("minx").item(0).firstChild.nodeValue;
document.getElementById("miny").innerHTML =
xml.getElementsByTagName("miny").item(0).firstChild.nodeValue;
document.getElementById("maxx").innerHTML =
xml.getElementsByTagName("maxx").item(0).firstChild.nodeValue;
document.getElementById("maxy").innerHTML =
xml.getElementsByTagName("maxy").item(0).firstChild.nodeValue;
document.getElementById("map-scale").innerHTML =
xml.getElementsByTagName("mapscale").item(0).firstChild.nodeValue;
```
Use case 1 is now complete.

Use case 2: AJAXResponseRenderer (Map Zooms application)

This use case shows how to accomplish the following tasks:

- Use a PhaseListener to change the state of a control
- Have the change update any other control that is part of the Web context
- Use the AJAXResponseRenderer object to help render the XML to update the controls on the browser page

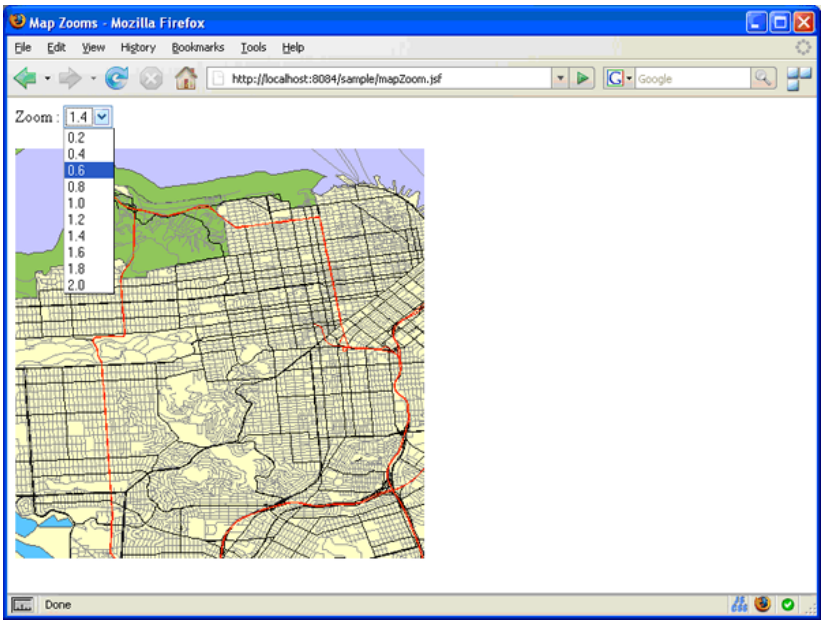
When running the Map Zooms application, selecting a zoom factor less than 1.0 zooms into the map, while a factor greater than 1.0 will zoom out on the map.

mapZoom.jsp

The mapZoom.jsp page declares the controls to be used in this application. This application contains a single map control. It uses a drop-down menu to zoom in or out of the map based on a specified zoom factor. The following code sample shows how to use the specified zoom factor.

[Java]
```
 < select id = "zoomFactor" onchange =

"sendMapZoomRequest(document.getElementById('zoomFactor').value);" >
< option
     value = "0.2" > 0.2 <  / option >  < option value = "0.4" > 0.4
<  / option >
     < option value = "0.6" > 0.6 <  / option >  < option value =
"0.8" > 0.8 <  /
     option >  < option value = "1.0" selected = "true" > 1.0 <  /
option >  <
     option value = "1.2" > 1.2 <  / option >  < option value = "1.4"
> 1.4 <  /
     option >  < option value = "1.6" > 1.6 <  / option >  < option
value = "1.8" >
     1.8 <  / option >  < option value = "2.0" > 2.0 <  / option >  <
/ select >
```
The following screen shot is of the Map Zooms application.



The JavaScript sendMapZoomRequest() function is responsible for sending the request to the server with the zoom factor selected. The param string contains the required mapZoom=mapZoom parameter so that com.esri.arcgis.sample.mapzoom.MapZoomPhaseListener can determine if it is responsible for processing the request. This is shown in the following code sample.

[Javascript]
```
var params = "mapZoom=mapZoom&formId=" + map.formId + "&mapId=" +
map.id +
    "&factor=" + factor + "&" +
EsriUtils.buildRequestParams(map.formId);
```
The callback function to process the XML response is the EsriControls.processPostBack() function. This function is the central JavaScript function that delegates the processing of the response XML to each of the controls on the page based on the list of registered tag handlers. This is shown in the following code sample.
[Java]
```
var xmlHttp = EsriUtils.sendAjaxRequest(url, params, true,
function(){
    EsriControls.processPostBack(xmlHttp);
}

);
```

MapZoomPhaseListener.java

MapZoomPhaseListener is responsible for processing the request, getting the map control, zooming in and out of the map based on the factor selected, and using the AJAXResponseRenderer object to render the XML response. This PhaseListener also executes in the Apply Request Values phase as shown in the following code sample.

[Java]
```
public PhaseId getPhaseId(){
    return PhaseId.APPLY_REQUEST_VALUES;
}
```
Determine if this PhaseListener is responsible for processing the request as shown in the following code sample.

[XML]
```
if (! “mapZoom”.equals((String) paramMap.get(“mapZoom”)))
  return;
```
As with MapInformationPhaseListener, MapZoomPhaseListener uses the form and map control IDs to get to the MapControl and the GIS business objects that back the MapControl—namely the WebMap and the WebContext—as shown in the following code sample.
[XML]
```
UIComponent form =
facesContext.getViewRoot().findComponent((String)paramMap.get(“formId
”));
MapControl mapControl = (MapControl) form.findComponent((String)
paramMap.get(“mapId”));
WebMap webMap = mapControl.getWebMap();
WebContext webContext = webMap.getWebContext();
```
Because AJAXResponseRenderer is used to update the map control on the page, an array of IDs must be created to determine the IDs of the controls that trigger the change to the application. In this case, it is the map, since the zoom operation is performed on the map. This is shown in the following code sample.

[Java]
```
Vector eventSources = new Vector();
eventSources.add(mapControl.getId());
```
Create an instance of the AJAXResponseRenderer class. When creating an instance of this object, pass in the faces context, the event sources, and the form that contain the controls you want to update as shown in the following code sample.

[Java]
```
AJAXResponseRenderer ajaxRenderer = new
AJAXResponseRenderer(facesContext,
    eventSources, form);
```
The AJAXResponseRenderer class is instantiated before any change is made to the GIS business objects. This is done because, when the AJAXResponseRenderer is initialized, it finds all the com.esri.adf.web.faces.renderkit.xml.ajax.AJAXRenderer classes listed in WEB-INF/ajax-renderers.xml. It iterates through the controls in the form, and, based on the class name returned by AJAXRenderer.getControlClass(), it calls and stores the object returned by calling AJAXRenderer.getOriginalState(). The original state represents the state of the control before it may have been affected by the change.
Execute the business logic of zooming in and out of the map control based on the selected factor as shown in the following code sample.
[Java]
```
WebExtent webExtent = webMap.getCurrentExtent();
webExtent.expand(Double.parseDouble((String)paramMap.get(FACTOR)));
webMap.setCurrentExtent(webExtent);
```
Call com.esri.adf.web.data.WebContext.refresh() to update all the com.esri.adf.web.data.WebContextObservers as shown in the following code sample.
[Java]
```
webContext.refresh();
```
Now that all the business objects have been updated based on the zoom operation on the map, use the AJAXResponseRenderer instance to render the XML response document back as shown in the following code sample.

[Java]
```
Document doc = ajaxRenderer.renderResponse(facesContext);
```
When calling AjaxRenderer.renderResponse, AJAXResponseRenderer again iterates through the controls in the form, calls AJAXRenderer.renderAjaxResponse, and passes appropriate parameters for building the XML document. The response is written and the faces context implementation is notified that the response is complete as shown in the following code sample.

[Java]
```
AJAXUtil.writeResponse(facesContext, doc);
facesContext.responseComplete();
```
Since, in the calling function, the callback function for the request was set to EsriControls.processPostBack, this function now parses the XML response, finds the tags within the root <response> tag, and calls the appropriate tag handlers to process the XML tag. For example, the <map> tag is processed by the EsriMap.updateAsync() function.

Use case 2 is now complete.