The AjaxTags Library: The Basics

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Java 5 or 6, etc. Spring/Hibernate coming soon.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.
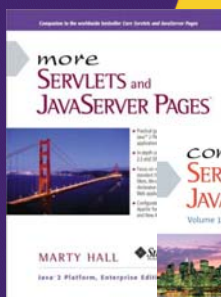
For live Ajax & GWT training, see training courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics
• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Spring, Hibernate, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details

Topics in This Section

- **Pros and cons of AjaxTags library**
- **Installing AjaxTags**
- **Using main components**
  - Links that trigger server-side resource and display results within current page
  - Autocompleting textfields
  - Populating textfields based on values in another textfield
  - Populating combobox based on selection in another combobox
  - Forms whose results are displayed inside current page
  - Tabbed panels

Overview and Installation

Overview of AjaxTags

- **Set of JSP custom tags to perform common Ajax operations**
  - One of the biggest bangs for the buck of all Java-related Ajax tools
    - Very low learning curve compared to other libraries
  - Built on top of script.aculo.us and Prototype
- **Pros**
  - Very easy to use
  - Moderately powerful
  - Can extend functionality moderately easily with post functions and prefunctions
- **Cons**
  - Moderately difficult to extend core tags
  - Future support and updates uncertain

Installation

- **Required Components**
  - A variety of JAR files (to go in WEB-INF/lib)
    - ajaxtags-1.2-xxx.jar
    - JSTL, Jakarta Commons, several others
  - Script.aculo.us and Prototype JavaScript files
    - To go in *WebContent*/scripts or similar location
- **Downloading**
  - Build your own
    - JAR files from http://ajaxtags.sourceforge.net/
      - But you have to dig for supporting JAR files
    - JavaScript from http://script.aculo.us/
  - Prebundled
    - Get ajaxtags-blank.zip from http://courses.coreservlets.com/Course-Materials/ajax.html

Documentation

- **AjaxTags core API**
  - http://ajaxtags.sourceforge.net/usage.html
- **AjaxTags Java API and advanced topics**
  - http://ajaxtags.sourceforge.net/advanced.html
- **AjaxTags source code**
  - http://sourceforge.net/project/showfiles.php?group_id=140499
- **Script.aculo.us docs**
  - http://wiki.script.aculo.us/scriptaculous/show/VisualEffects
  - http://wiki.script.aculo.us/scriptaculous/show/Usage
- **Prototype docs**
  - http://prototypejs.org/api

Most Important Tags

- **ajax:anchors**
  - Link that specifies a URL; result pops up in current page
    - In specified section (usually a div or span)
- **ajax:autocomplete**
  - Textfield with dropdown giving completion options
    - Completion options come from server
- **ajax:updateField**
  - Field whose value is used to populate other field(s)
- **ajax:select**
  - Pair of linked combo (dropdown) boxes.
    - When first one changes, result is sent to server and used to compute values for second
- **ajax:htmlContent**
  - Button, link, or other element that invokes server-side resource and displays result in current page
- **ajax:tabPanel**
  - Set of tabbed panels where contents of each panel comes from a different server-side resource

Tags Examples: index.jsp

```
<!DOCTYPE ...>
<%@ taglib uri="http://ajaxtags.org/tags/ajax"
          prefix="ajax" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type"
      content="text/html;charset=UTF-8"" />
<jsp:include page="/WEB-INF/includes/header.jsp"/>
<title>AjaxTags Examples</title>
</head>
<body>
...
</body>
</html>
```

Tags Examples: header.jsp

```
<% request.setAttribute("contextPath",
                        request.getContextPath()); %>
<script src="${contextPath}/scripts/prototype-1.4.0.js"
        type="text/javascript"></script>
<script src="${contextPath}/scripts/scriptaculous.js"
        type="text/javascript"></script>
<script src="${contextPath}/scripts/overlibmws.js"
        type="text/javascript"></script>
<script src="${contextPath}/scripts/ajaxtags-1.2-beta2.js"
        type="text/javascript"></script>
<link rel="stylesheet"
      href="${contextPath}/css/styles.css"
      type="text/css"/>
```

ajax:anchors

Tag Usage

- ## General usage
  - Wrap around a hypertext link. When link clicked, result pops up in a designated region.
- ## Tag attributes
  - target
    - id of the region where result should be displayed. Usually an initially-empty div or span element
  - parameters
    - Comma separated list of parameters that should be sent to server-side resource listed in the hypertext link.
    - Example:
      <ajax:anchors ...parameters="name1={id1},name2={id2}">
      - Value in curly braces is the <u>id</u> of a textfield or other form entry
      - Note that there is no $ -- it is {id}, not ${id}

JSP Example

```
<fieldset>
  <legend>ajax:anchors</legend>
  <ajax:anchors target="time">
    <a href="${contextPath}/show-time.jsp">
    Show current time</a>
  </ajax:anchors>
    <span id="time"></span>
</fieldset>
```

- **Notes**
  - contextPath variable defined in header as request.getContextPath() and stored in request scope
  - Top of page used @taglib to enable ajax: tags
  - Attached style sheet used to style legend element

Style Sheet Entries (in styles.css)

```
legend {
  font-weight: bold;
  color: black;
  background-color: white;
  border: 1px solid #cccccc;
  padding: 4px 2px;
}
```

Server-Side Code

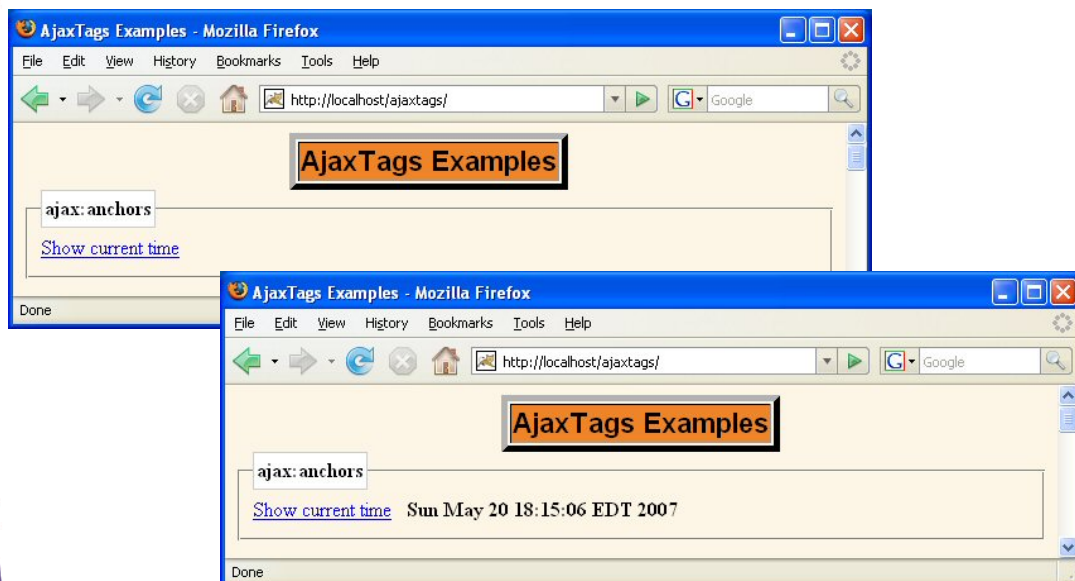- **show-time.jsp**

```
<b><%= new java.util.Date() %></b>
```

Results

ajax:autocomplete

Tag Usage

- **General usage**
  - Place *below* form. Designate a server-side resource that is called each time textfield changes. Resource returns list that is displayed in dropdown.
- **Tag attributes**
  - source
    - id of the textfield where user will be typing
  - target
    - id of the textfield where result from dropdown displayed
    - Usually same id as target, but see advanced usage section
  - baseUrl
    - Address of server-side resource
  - parameters
    - Comma separated list of parameters
  - className
    - CSS class name to apply to dropdown box. Dropdown will be formatted as a list, so CSS should suppress bullets and use absolute position
  - minimumCharacters
    - Number of chars in textfield before server-side resource triggered

JSP Example

```
<fieldset>
   <legend>ajax:autocomplete</legend>
   <form>
      <label for="language">Programming language:</label>
      <input type="text" name="language" id="language"/>
   </form>
   <ajax:autocomplete
      source="language"
      target="language"
      baseUrl="${contextPath}/language-completer.ajax"
      parameters="language={language}"
      className="autocomplete"
      minimumCharacters="1"/>
</fieldset>
```

Form closed <u>before</u> ajax:autocomplete

Textfield to monitor

Field that should have dropdown

url-pattern
of servlet

Param name server sees.   Actual element that has input.

Style Sheet Entries

```
.autocomplete {
  position: absolute;
  color: #333333;
  background-color: #ffffff;
  border: 1px solid #666666;
  font-family: Arial;
  overflow: hidden;
}
.autocomplete ul {
  padding: 0;
  margin: 0;
  list-style: none;
  overflow: auto;
}
```

```
.autocomplete li {
  display: block;
  white-space: nowrap;
  cursor: pointer;
  margin: 0px;
  padding-left: 5px;
  padding-right: 5px;
  border: 1px solid #ffffff;
}
.autocomplete li.selected {
  background-color: #cceeff;
  border-top: 1px solid
                   #99bbcc;
  border-bottom: 1px solid
                   #99bbcc;
}
```

Server-Side Code: Return Value

- **Need to create servlet that returns list:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ajax-response>
  <response>
    <item>
      <name>display val 1</name>
      <value>selection val 1</value>
    </item>
    <item>
      <name>display val 2</name>
      <value>selection val 2</value>
    </item>
    ...
  </response>
</ajax-response>
```

Values to display in dropdown

Values to insert when value
from dropdown selected
(often identical to display val)

Creating ajax-response List: Shortcut

- **Extend BaseAjaxServlet**
  - In package org.ajaxtags.servlets
- **Override getXmlContent**
  - Takes same arguments as doGet or doPost
- **Create an AjaxXmlBuilder**
  - In package org.ajaxtags.helpers
- **Add items to list**
  - builder.addItem("displayVal", "selectionVal")
    - Display val and selection val the same for most cases, but see advanced usage section
- **Turn builder into XML list and return it**
  - return(builder.toString())

Server-Side Code

```
import javax.servlet.http.*;
import org.ajaxtags.helpers.*;
import org.ajaxtags.servlets.*;

public class LanguageCompleter extends BaseAjaxServlet {
  // 50 most popular programming languages, listed in order.
  // From http://www.tiobe.com/tpci.htm
  private static final String languages =
    "Java,C,C++,PHP,Visual Basic,Perl,Python,C#,...";
  private static final String[] languageNames =
    languages.split(",");

  @Override
  public String getXmlContent(HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {
    String languagePrefix = request.getParameter("language");
    String languageList = makeLanguageList(languagePrefix);
    return(languageList);
  }
```

Server-Side Code

```
  private String makeLanguageList(String languagePrefix) {
    AjaxXmlBuilder builder = new AjaxXmlBuilder();
    for(String language: languageNames) {
      if(language.toUpperCase().startsWith
                        (languagePrefix.toUpperCase())) {
        builder.addItem(language, language);
      }
    }
    return(builder.toString());
  }
}
```

web.xml

```
...
  <servlet>
    <servlet-name>LanguageCompleter</servlet-name>
    <servlet-class>
      coreservlets.LanguageCompleter
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LanguageCompleter</servlet-name>
    <url-pattern>/language-completer.ajax</url-pattern>
  </servlet-mapping>
...
```
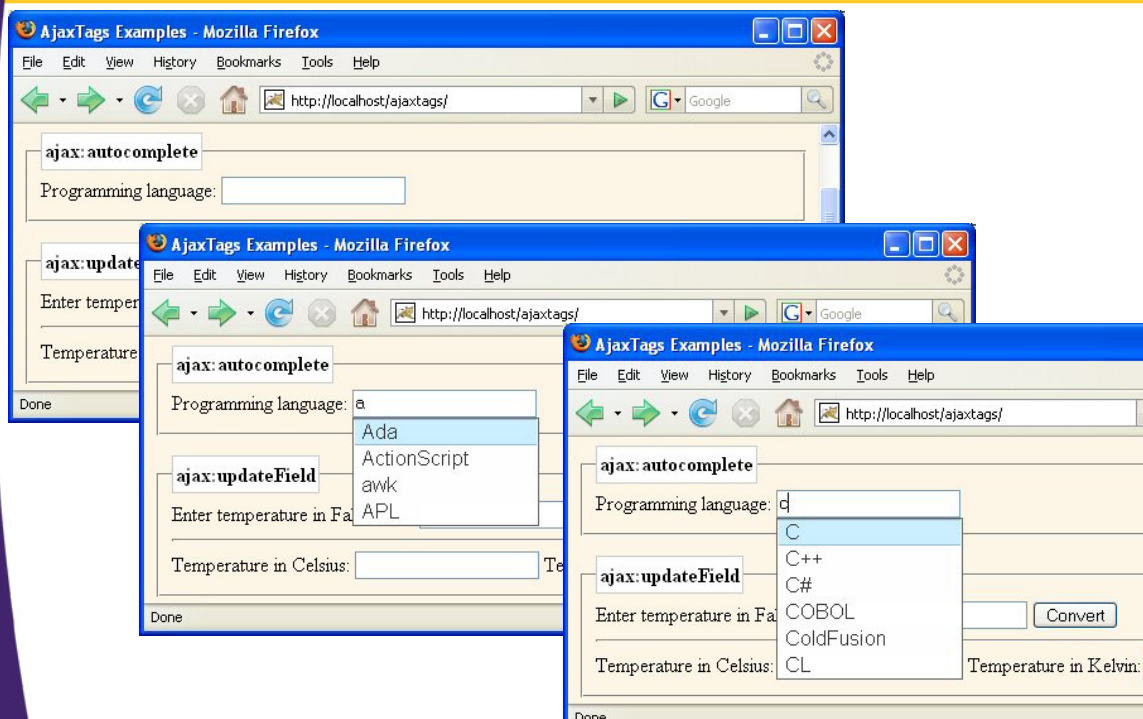
URL must match baseUrl. Name is arbitrary.
End user will never see URL.

Results

ajax:updateField

Tag Usage

- **General usage**
  - Place *below* form. Designate a server-side resource. Resource returns values that are inserted into other textfields.
- **Tag attributes**
  - source
    - id of the textfield containing initial value
  - target
    - id of the textfield where result from server (derived value) will go
      - Can supply a comma-separated list of ids
  - baseUrl
    - Address of server-side resource
      - If one result textfield, resource should return a single string
      - If multiple result textfields, resource should return a list
  - action
    - id of button or other element that will trigger submission
  - parameters
    - Comma separated list of parameters
  - parser
    - Omit if you return a string for use in a single textfield
    - Use parser="new ResponseXmlParser()" if you return a list for use in multiple textfields

JSP Example

```
<fieldset>
  <legend>ajax:updateField</legend>
  <form>
    <label for="f">Enter temperature in Fahrenheit:</label>
    <input type="text" id="f"/>
    <input type="button" id="convertButton" value="Convert"/>
    <hr width="500" align="left"/>
    <label for="c">Temperature in Celsius:</label>
    <input type="text" id="c"/>
    <label for="k">Temperature in Kelvin:</label>
    <input type="text" id="k"/>
  </form>
  <ajax:updateField
    source="f"
    target="c,k"
    baseUrl="${contextPath}/temperature-converter.ajax"
    action="convertButton"
    parameters="f={f}"
    parser="new ResponseXmlParser()"/>
</fieldset>
```

Server-Side Code: Return Value

• **If you have a single textfield for the result**
– And the default parser

```
public String getXmlContent(HttpServletRequest request,
                            HttpServletResponse response)
    throws Exception {
  String result = getResult(...);
  return(result);
}
```

Server-Side Code: Return Value

- **If you have multiple textfields for results**
  - And set the parser to ResponseXmlParser

```
public String getXmlContent(HttpServletRequest request,
                            HttpServletResponse response)
    throws Exception {
  AjaxXmlBuilder builder = new AjaxXmlBuilder();
  builder.addItem("name1", "val for textfield1");
  builder.addItem("name1", "val for textfield2");
  return(builder.toString());
}
```

Server-Side Code

```
public String getXmlContent(HttpServletRequest request,
                            HttpServletResponse response)
    throws Exception {
  String fString = request.getParameter("f");
  double fTemp = -500;
  try {
    fTemp = Double.parseDouble(fString);
  } catch(NumberFormatException nfe) {}
  String degreesC = "Illegal Temp";
  String degreesK = degreesC;
  if (fTemp >= -459.4) {
    double cTemp = (fTemp - 32)*(5.0/9.0);
    double kTemp = cTemp + 273;
    degreesC = String.format("%.2f", cTemp);
    degreesK = String.format("%.2f", kTemp);
  }
  AjaxXmlBuilder builder = new AjaxXmlBuilder();
  builder.addItem("c", degreesC);
  builder.addItem("k", degreesK);
  return(builder.toString());
}
```

web.xml

```
...
  <servlet>
    <servlet-name>TemperatureConverter</servlet-name>
    <servlet-class>
      coreservlets.TemperatureConverter
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TemperatureConverter</servlet-name>
    <url-pattern>/temperature-converter.ajax</url-pattern>
  </servlet-mapping>
...
```
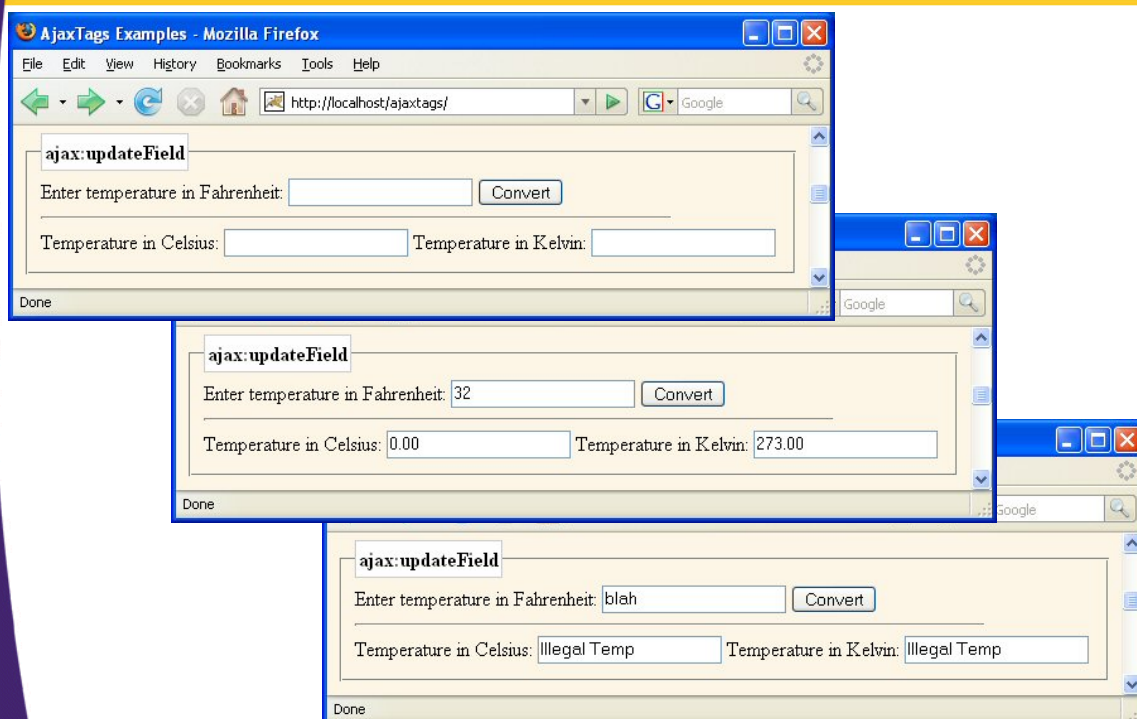
Results

ajax:select

Tag Usage

- **General usage**
  - Place *below* form. When first dropdown (combobox) changes, server-side resource is invoked and values placed in second dropdown.
- **Tag attributes**
  - source
    - id of the dropdown box containing initial value
  - target
    - id of the dropdown where results from server will go
  - baseUrl
    - Address of server-side resource
      - Resource should return a list
  - parameters
    - Comma separated list of parameters

JSP Example

```
<fieldset>
  <legend>ajax:select</legend>
  <form>
    <label for="state">State:</label>
    <select id="state">
      <option value="">Select State</option>
      <option value="Maryland">Maryland</option>
      <option value="Virginia">Virginia</option>
      <option value="Pennsylvania">Pennsylvania</option>
      <option value="New Jersey">New Jersey</option>
      <option value="New York">New York</option>
    </select>
    <label for="city">City:</label>
    <select id="city" disabled="disabled">
      <option value="">Select City</option>
    </select>
  </form>
  <ajax:select
    source="state"
    target="city"
    baseUrl="${contextPath}/city-finder.ajax"
    parameters="state={state}" />
</fieldset>
```

Have "dummy" value at top. Otherwise no event will fire if first state (Maryland) selected.

Second combobox should be initially disabled. AjaxTags will enable it when value inserted.

Server-Side Code: Return Value

- **Build a list of results**
  – Item name and item value identical in this case

```
public String getXmlContent(HttpServletRequest request,
                            HttpServletResponse response)
    throws Exception {
  findDataBasedOnParameterFromFirstCombobox(...);
  AjaxXmlBuilder builder = new AjaxXmlBuilder();
  builder.addItem(val1, val1);
  builder.addItem(val2, val2);
  ...
  return(builder.toString());
}
```

Server-Side Code

```
public class CityFinder extends BaseAjaxServlet {
  private Map<String,String> cityMap;

  @Override
  public String getXmlContent(HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {
    String state = request.getParameter("state");
    String cityList = cityMap.get(state);
    if (cityList == null) {
      return("");
    } else {
      return(cityList);
    }
  }
```

Server-Side Code (Continued)

```
  @Override
  public void init() {
    cityMap = new HashMap<String,String>();
    for(StateInfo state: StateInfo.getNearbyStates()) {
      cityMap.put(state.getStateName(),
                  makeCityList(state.getCities()));
    }
  }

  private String makeCityList(CityInfo[] cities) {
    AjaxXmlBuilder builder = new AjaxXmlBuilder();
    for(CityInfo city: cities) {
      builder.addItem(city.getCityName(),
                      city.getCityName());
    }
    return(builder.toString());
  }
}
```

Server-Side Code
(StateInfo Helper Class)

```java
public class StateInfo {
  private String stateName;
  private CityInfo[] cities;

  public StateInfo(String stateName, CityInfo...cities) {
    setStateName(stateName);
    setCities(cities);
  }

  public String getStateName() {
    return(stateName);
  }

  public void setStateName(String stateName) {
    this.stateName = stateName;
  }

  public CityInfo[] getCities() {
    return(cities);
  }

  public void setCities(CityInfo[] cities) {
    this.cities = cities;
  }
```

Server-Side Code
(StateInfo Helper Class Continued)

```java
  private static StateInfo[] nearbyStates =
    { new StateInfo("Maryland",
                    new CityInfo("Baltimore", 635815),
                    new CityInfo("Frederick", 57907),
                    new CityInfo("Gaithersburg", 57698),
                    new CityInfo("Rockville", 57402)),
      new StateInfo("Virginia",
                    new CityInfo("Virginia Beach", 438415),
                    new CityInfo("Norfolk", 231954),
                    new CityInfo("Chesapeake", 218968),
                    new CityInfo("Arlington", 195965)),
      new StateInfo("Pennsylvania",
                    new CityInfo("Philadelphia", 1463281),
                    new CityInfo("Pittsburgh", 316718),
                    new CityInfo("Allentown", 106992),
                    new CityInfo("Erie", 102612)),
      ... };

  public static StateInfo[] getNearbyStates() {
    return(nearbyStates);
  }
```

Server-Side Code
(CityInfo Helper Class)

```
public class CityInfo {
  private String cityName;
  private int population;

  public CityInfo(String cityName, int population) {
    setCityName(cityName);
    setPopulation(population);
  }

  public String getCityName() {
    return(cityName);
  }
  public void setCityName(String cityName) {
    this.cityName = cityName;
  }

  public int getPopulation() {
    return(population);
  }
  public void setPopulation(int population) {
    this.population = population;
  }
}
```

web.xml

```
...
  <servlet>
    <servlet-name>CityFinder</servlet-name>
    <servlet-class>
      coreservlets.CityFinder
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CityFinder</servlet-name>
    <url-pattern>/city-finder.ajax</url-pattern>
  </servlet-mapping>
...
```
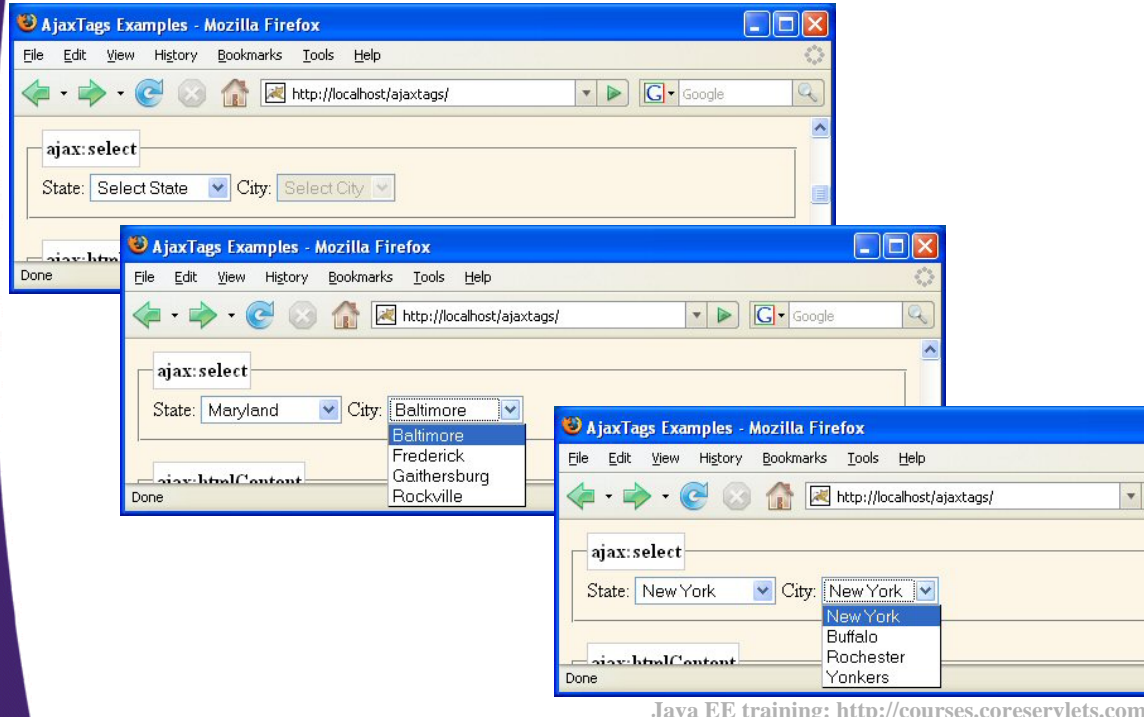
Results

---

ajax:htmlContent

Tag Usage

- **General usage**
  - Place *below* form. When source element clicked, server-side resource is invoked and values placed inside html area.
- **Tag attributes**
  - source
    - id of the button or other element that will trigger submission
  - target
    - id of the html element where results from server will go
      - Usually a div or span element
  - baseUrl
    - Address of server-side resource
    - Resource should return regular HTML, *not* XML
  - parameters
    - Comma separated list of parameters

JSP Example

```
<fieldset>
  <legend>ajax:htmlContent</legend>
  <form>
    <label for="state2">State:</label>
    <select id="state2">
      <option value="">Select State</option>
      <option value="Maryland">Maryland</option>
      <option value="Virginia">Virginia</option>
      <option value="Pennsylvania">Pennsylvania</option>
      <option value="New Jersey">New Jersey</option>
      <option value="New York">New York</option>
    </select>
    <label for="city2">City:</label>
    <select id="city2" disabled="disabled">
      <option value="">Select City</option>
    </select>
    <input type="button"
           value="Show Population" id="button"/>
      <span id="population"></span>
  </form>
```

Earlier dropdowns called "state" and "city". id's in page must be unique.

JSP Example (Continued)

```
<ajax:select
  baseUrl="${contextPath}/city-finder.ajax"
  source="state2"
  target="city2"
  parameters="state={state2}"/>
<ajax:htmlContent
    baseUrl="${contextPath}/population-finder.ajax"
    source="button"
    target="population"
    parameters="state={state2},city={city2}"/>
</fieldset>
```

To populate second dropdown.

To compute population once both
dropdowns are used and button pressed.

Server-Side Code: Return Value

- **Return regular HTML**
  – Not XML
  – No need for AjaxXmlBuilder

```
public String getXmlContent(HttpServletRequest request,
                            HttpServletResponse response)
    throws Exception {
  findDataBasedOnParameters(...);
  String result =
    String.format("<html-tag>...%s</html-tag>", data);
  ...
  return(result);
}
```

Server-Side Code

```java
public class PopulationFinder extends BaseAjaxServlet {
  @Override
  public String getXmlContent(HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {
    String state = request.getParameter("state");
    String city = request.getParameter("city");
    int population = findPopulation(state, city);
    String populationString;
    if (population == 0) {
      populationString =
        String.format
          ("<b>Don't know the population of %s.</b>", city);
    } else {
      populationString =
        String.format("<b>Population of %s, %s is %,d.</b>",
                      city, state, population);
    }
    return(populationString);
  }
```

Server-Side Code (Continued)

```java
  private int findPopulation(String state, String city) {
    for(StateInfo stateInformation:
        StateInfo.getNearbyStates()) {
      if (stateInformation.getStateName().equals(state)) {
        for(CityInfo cityInformation:
            stateInformation.getCities()) {
          if (cityInformation.getCityName().equals(city)) {
            return(cityInformation.getPopulation());
          }
        }
      }
    }
    return(0);
  }
}
```

web.xml

```
...
  <servlet>
    <servlet-name>PopulationFinder</servlet-name>
    <servlet-class>
      coreservlets.PopulationFinder
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>PopulationFinder</servlet-name>
    <url-pattern>/population-finder.ajax</url-pattern>
  </servlet-mapping>
...
```
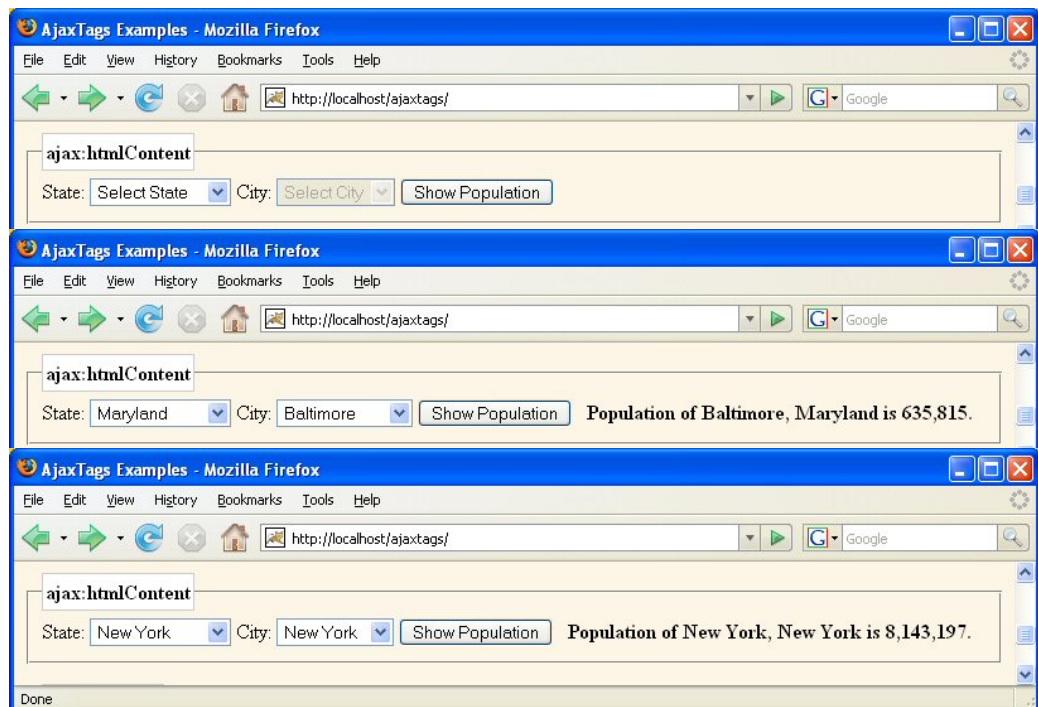
Results

ajax:tabPanel (and ajax:tab)

Tag Usage

- **General usage**
  - Not necessarily any explicit form. Each tab populated by data from a different URL.
- **Tag attributes**
  - ajax:tabPanel
    - panelStyleID, contentStyleId
      - ids for generated divs
    - panelStyleClass, contentStyleClass, currentStyleClass
      - CSS style names for unselected tabs, tab content, selected tab
  - ajax:tab
    - caption
      - Text to show on tab
    - baseUrl
      - Address of server-side resource (resource should return HTML)
    - parameters
      - Comma separated list of parameters
    - defaultTab (true or false)
      - Indicates if tab is initially selected. You *must* specify true for one tab.

JSP Example

```
<fieldset>
  <legend>ajax:tabPanel</legend>
  <h2>Largest Cities in Selected Northeast States</h2>
  <div class="tabPanelWrapper">
    <ajax:tabPanel
      panelStyleId="panel"
      contentStyleId="content"
      panelStyleClass="tabPanel"
      contentStyleClass="tabContent"
      currentStyleClass="currentTab">
      <ajax:tab
        caption="Maryland"
        baseUrl="${contextPath}/population-finder.ajax"
        parameters="state=Maryland,city=Baltimore"
        defaultTab="true"/>
      <ajax:tab
        caption="Virginia"
        baseUrl="${contextPath}/population-finder.ajax"
        parameters="state=Virginia,city=Virginia Beach"/>
      ...
    </ajax:tabPanel>
  </div>
</fieldset>
```

Wrap entire tabbed panel in a div with a fixed width.

• Unselected tabs
• Panel contents (should have border)
• Selected tab

Style Sheet Entries

```
.tabPanelWrapper {
  width: 450px;
}
.tabPanel {
  border-bottom: 1px solid
                        #cccccc;
  margin: 0;
  ...
}
.tabPanel ul, .tabPanel li {
  display: inline;
  list-style-type: none;
  ...
}
.tabPanel a:link,
    .tabPanel a:visited {
  background: #E8EBF0;
  border: 1px solid #ccc;
  color: #666;
  ...
}
```

```
.tabContent {
  background: #ffffff;
  border: 1px solid #cccccc;
  border-top: none;
  clear: both;
  margin: 0px;
  padding: 15px;
}
.tabPanel a:link.currentTab,
.tabPanel a:visited.currentTab {
  background: #ffffff;
  border-bottom: 1px solid
                        #ffffff;
  color: #000000;
}
```

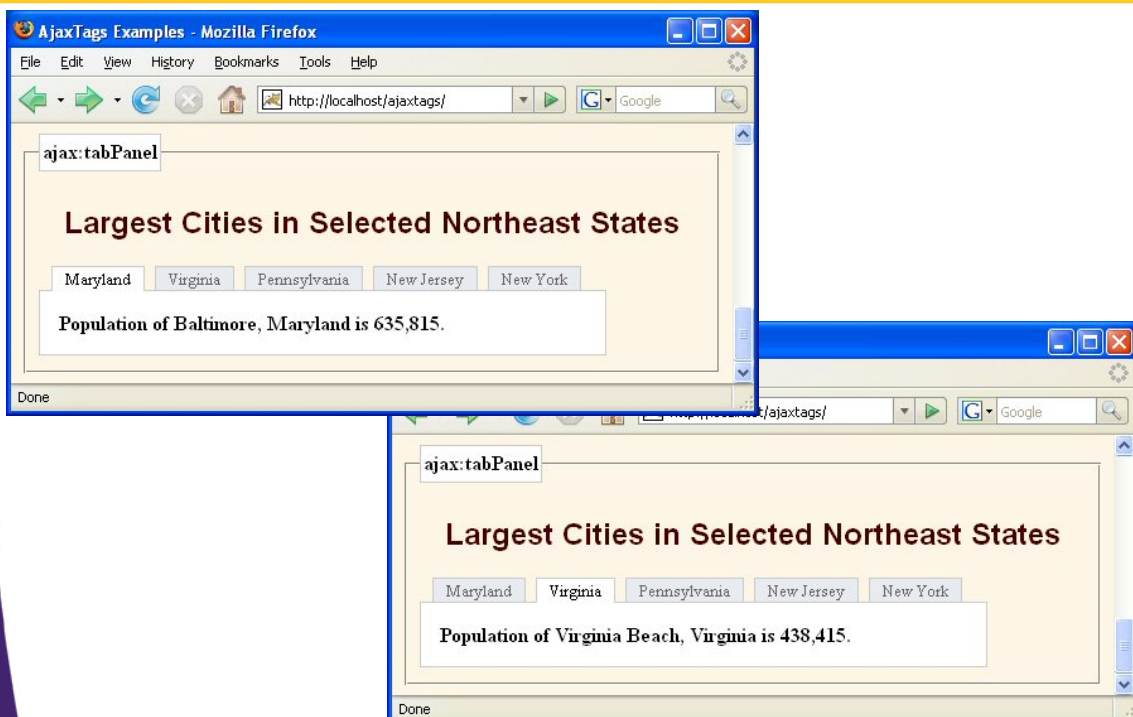# Server-Side Code

- **Same PopulationFinder as previous examples**
  - Given a state and a city, returns the population in an HTML (not XML) string

Results

Other Capabilities