Advanced JavaServer™ Faces Technology Custom Component Development

Chris Schalk

Principal Product Manager/Technology Evangelist
Oracle Corporation
www.oracle.com

TS-3187

Goal

Learn how to build a variety of custom JavaServer Faces based components ranging from simple to rich client architectures

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

The “Basics”

Of custom JavaServer Faces based Component development

- JavaServer Faces based UI Components consists of these “moving parts”
  - UIComponent class
    - Provides the behavior of the component
  - Renderer class
    - Rendering class that display component to client
    - Is optional, rendering can be in UIComponent as well
  - UIComponentTag Class
    - Tag handler for using UI component in JavaServer Pages™ (JSP™) technology
    - Associates Renderer with UIComponent
    - Is also optional, only for use with JSP based clients
  - Tag library descriptor (TLD)
    - Optional, again only for JSP technology

The "Basics"

Of custom JavaServer Faces based Component development

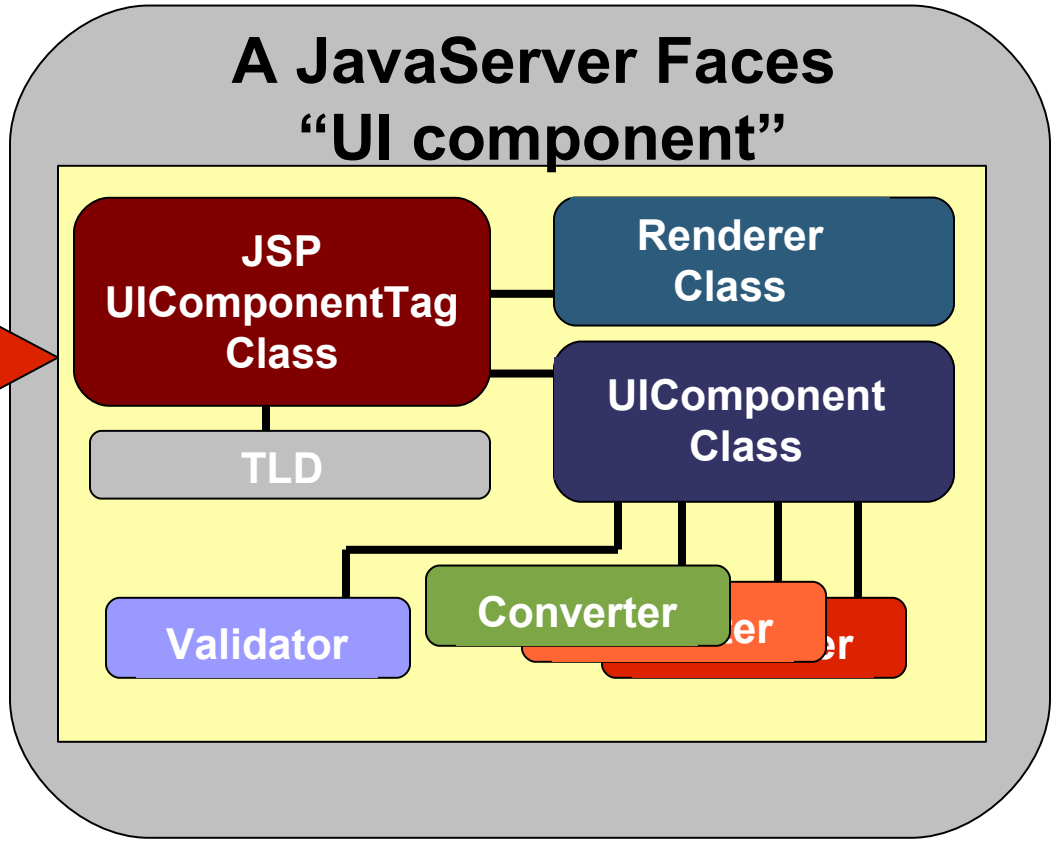
- JavaServer Faces based UI Components can also *contain* other non-visual components such as:
  - Validators
    - Validates incoming data
  - Converters
    - Converts submitted String value to custom server-side value (Ex: CreditCard…)
  - And so on…

The “Basics”

Of custom JavaServer Faces based Component development

```
<f:view>
<html>
<body>
<h:form>
  <jcr:mywidget>

</h:form>
</body>
</html>
</f:view>
```

Usage in JSP technology



UI Component Elements

A UIComponent Code Sample

```
public class HtmlHelloWorld extends UIComponentBase {

  public String getFamily(){
    return null;
  }

  public void encodeBegin(FacesContext context) throws
IOException {
    String hellomsg =
(String)getAttributes().get("hellomsg");

    ResponseWriter writer = context.getResponseWriter();
    writer.startElement("div", this);
    writer.writeAttribute("style", "color : red", null );
    writer.writeText(hellomsg, null);
    writer.endElement("div");
  }
}
```

A UIComponent Structure

```
Public class HtmlHelloWorld extends UIComponentBase

   Public String getFamily(){ ...}

   Public void encodeBegin(){ ...}

   Public void encodeChildren(){ ...}

   Public void encodeEnd(){ ...}

   Public void decode(){ ...}

       ...
```

A UIComponentTag Sample

```
public class HtmlHelloInputTag extends UIComponentTag {

  public String getComponentType(){
    return "HtmlHelloInput";
  }
  public String getRendererType(){
    return null;
  }
  protected void setProperties(UIComponent component){
    super.setProperties(component);
    …
  }
  public void release(){
    super.release();
  }
}
```

A Renderer Sample

```
public class HtmlInputDateRenderer extends Renderer {

  public HtmlInputDateRenderer() {
      super();
  }

  public void encodeBegin(FacesContext context,
                          UIComponent component) throws
IOException {
    …
  }

  public void decode(FacesContext context, UIComponent
component) {
    …
  }
}
```

DEMO

HtmlHelloworld and HtmlInputDate—Some first custom components

How to Register Renderkits and Renderer Families

- Registered Faces-config.xml
- Allows for association of component-family with renderer types

How to Register Renderkits and Renderer Families

```
<render-kit>
  <renderer>
    <component-family>InputDateFamily</component-family>
    <renderer-type>WmlInputDateRenderer</renderer-type>
    <renderer-class>
    com.jsfcompref.components.component.WmlInputDateRenderer
    </renderer-class>
  </renderer>
  <renderer>
    <component-family>InputDateFamily</component-family>
    <renderer-type>HtmlInputDateRenderer</renderer-type>
    <renderer-class>
    com.jsfcompref.components.component.HtmlInputDateRenderer
    </renderer-class>
  </renderer>
</render-kit>
```

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

Key things to Know When Building Custom JavaServer Faces Based Components

Thread Safety

- When building renderers should not use instance variables as they are not threadsafe
- Can place variables in rendering methods instead

Key things to Know When Building Custom JavaServer Faces Based Components

Tag Coding Details

- In general, you shouldn't place client specific code in your JSP Tag Class
- Tag Class properties should always be initialized with null values
  - Do not set default values in JSP Tag class
  - Set default values in renderer code
- Tag code should only have setters (no getters)

Key things to Know When Building Custom JavaServer Faces Based Components

Component Naming Guidelines

- Should use hierarchical naming for registered components in faces-config.
- Should convey rendering technology along with component type
  - For example: **HtmlInputDate**

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

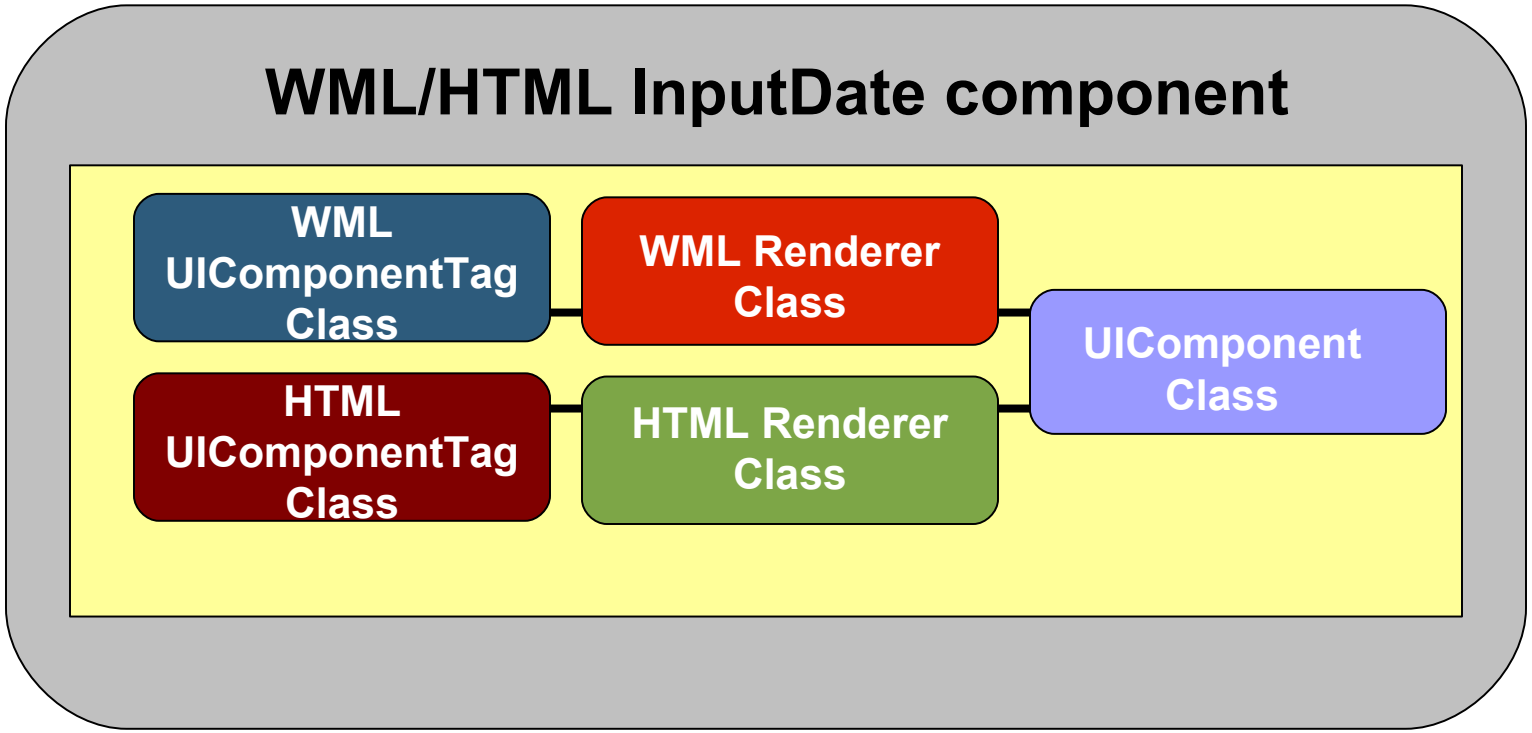
Rendering in Different Markups

- In addition to HTML, Faces renderers can render in any Markup such as WML, XML, SVG or even binary

An InputDate Component for WML

- WmlInputDate a variation of the HtmlInputDate component that renders WML instead
  - Same UI Component, but uses different renderer

HtmlInputDate and WmlInputDate Component Architecture



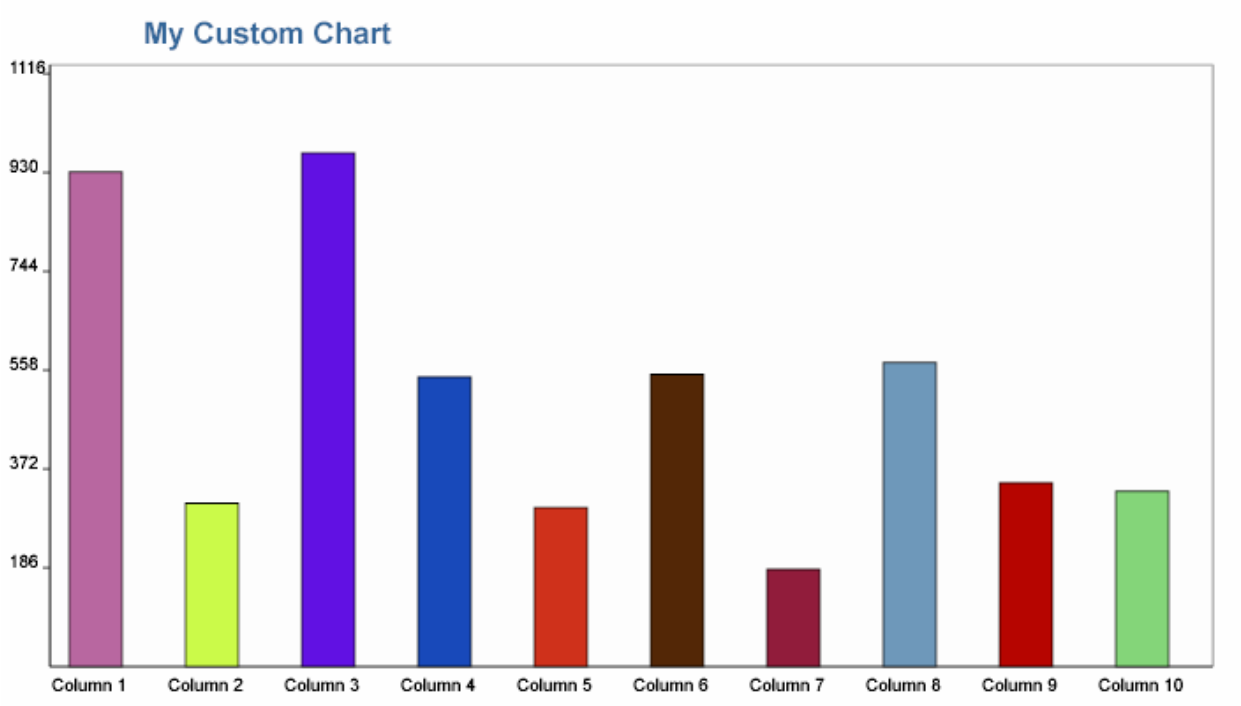
UI Component Elements

DEMO

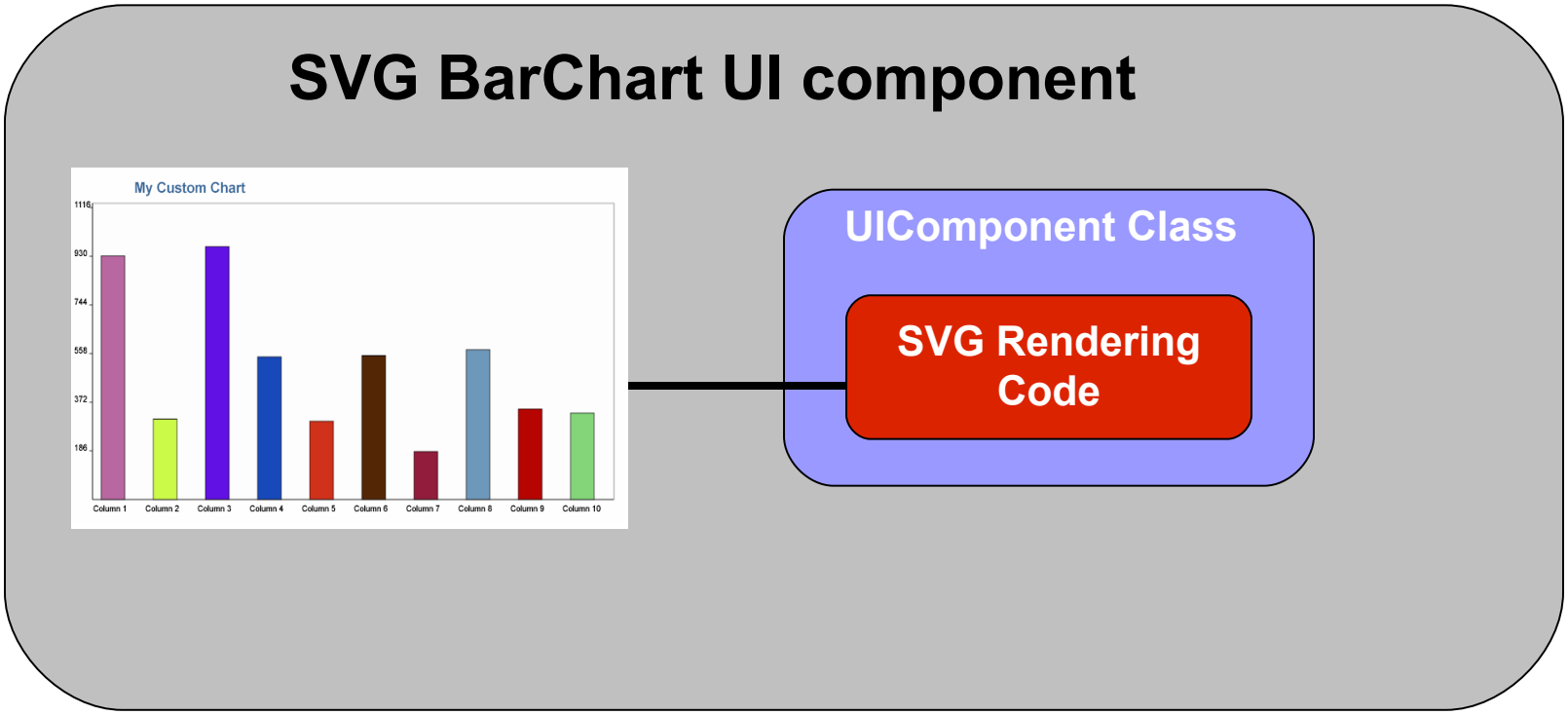
WMLInputDate —A WML InputDate Variation

Rendering SVG for Charts

- Can render SVG markup to build chart components



SVG Bar Chart Component Architecture



UI Component Elements

DEMO

An SVG Charting Component

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

Making Components Come Alive

- General shift towards Rich Client Architecture
- Realization that most browsers can support a much richer user interface in a consistent manner
- No longer a need to do the old-fashioned full page refresh for each interaction
- Ajax provides an easy way to extract data from server without page refresh. Hottest thing!

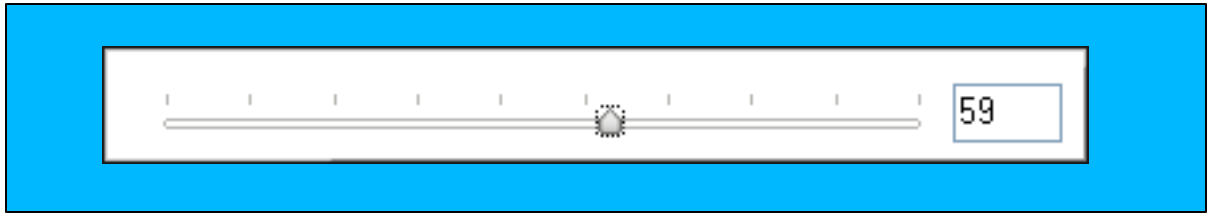
Making Components Come Alive

- The more general definition of AJAX: includes both rich client architecture as well as XMLHttpRequest
- A first rich client example (without XMLHttpRequest)
  - A slider component

A Slider JavaServer Faces Based Component

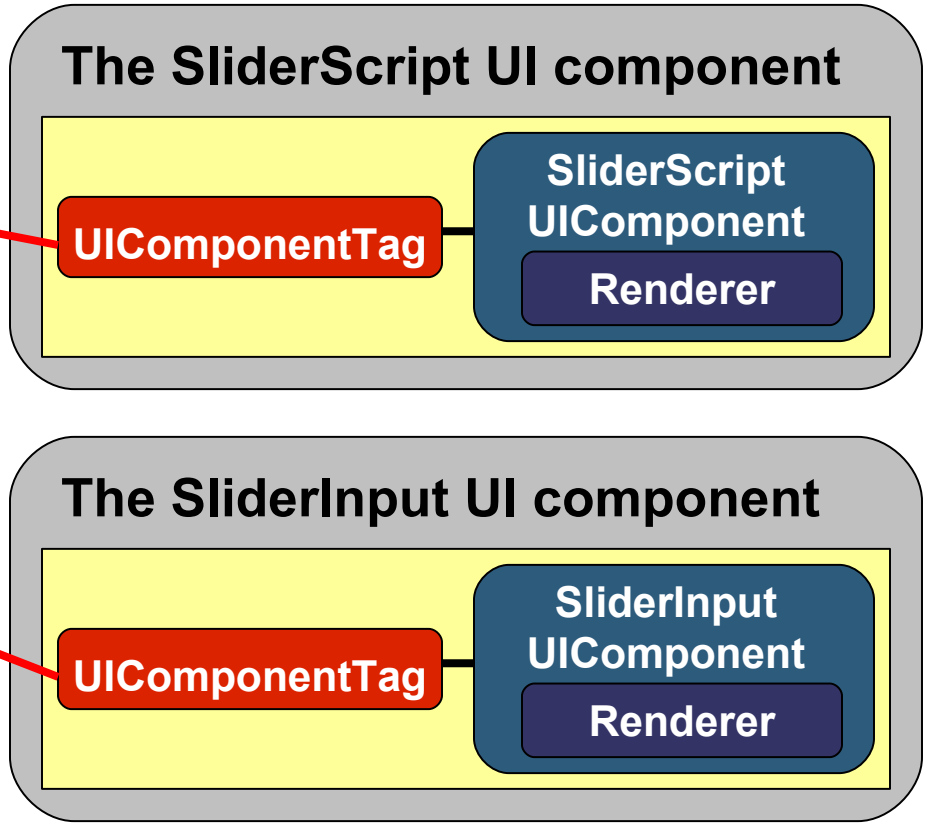
- Provides user with ability to manipulate a slider without incurring Http transaction.
- Uses JavaScript™ technology

- A first rich client example (without XMLHttpRequest)
  - A slider component



Slider JavaServer Faces Component Architecture

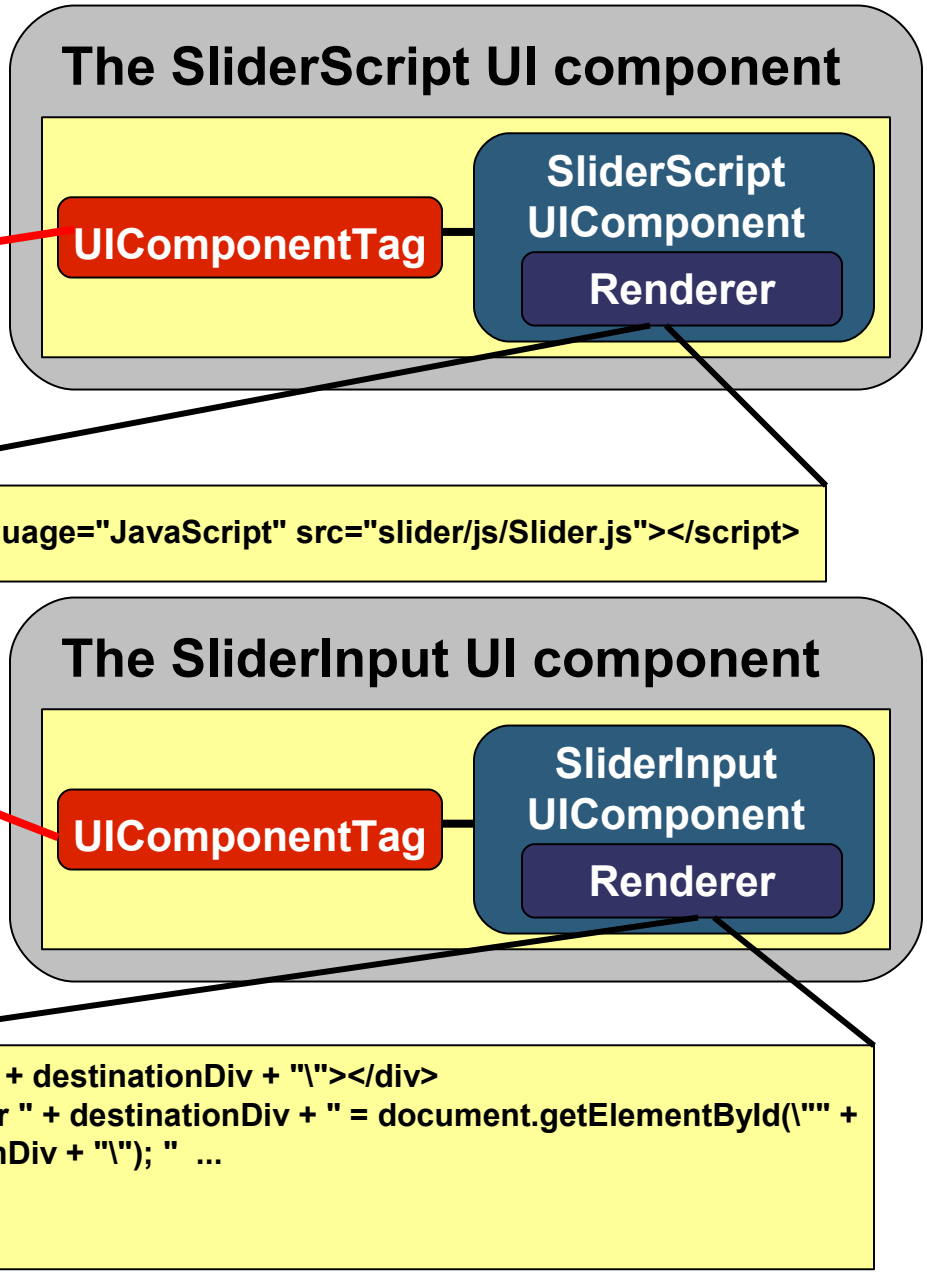
```
<f:view>
<html>
<head>
  <jcr:sliderscript />
</head>
<body>
<h:form>
…
  <jcr:slider … />

</h:form>
</body>
</html>
</f:view>
```



Usage in JSP technology

```
<f:view>
<html>
<head>
  <jcr:sliderscript />
</head>
<body>
<h:form>
…
  <jcr:slider … />

</h:form>
</body>
</html>
</f:view>
```

Usage in JSP technology



Strategies for Rendering JavaScript Code

- Can simply render JavaScript based inclusion tag
  - Ex: <script language="JavaScript" src="slider/js/Slider.js"></script>
  - Ideal for static JavaScript code
- Can render JavaScript code dynamically
  - Sometimes needed when JavaScript code must be “made to order”
- Both strategies used for Slider component

DEMO

Building a JavaServer Faces Based Slider Component

Agenda

The Basics

Rendering Strategies

Key things to know

Rendering in different Markups

Building Rich Client components

Strategies for integrating AJAX into JavaServer Faces technology

Summary

Adding AJAX to Your Custom Components

- AJAX (Asynchronous JavaScript™ technology and XML) allows for independent Web requests that don't necessitate full page refresh
- Provides for a much richer user interface
- JavaScript XMLHttpRequest object must be used by client side JavaScript code

Using XMLHttpRequest

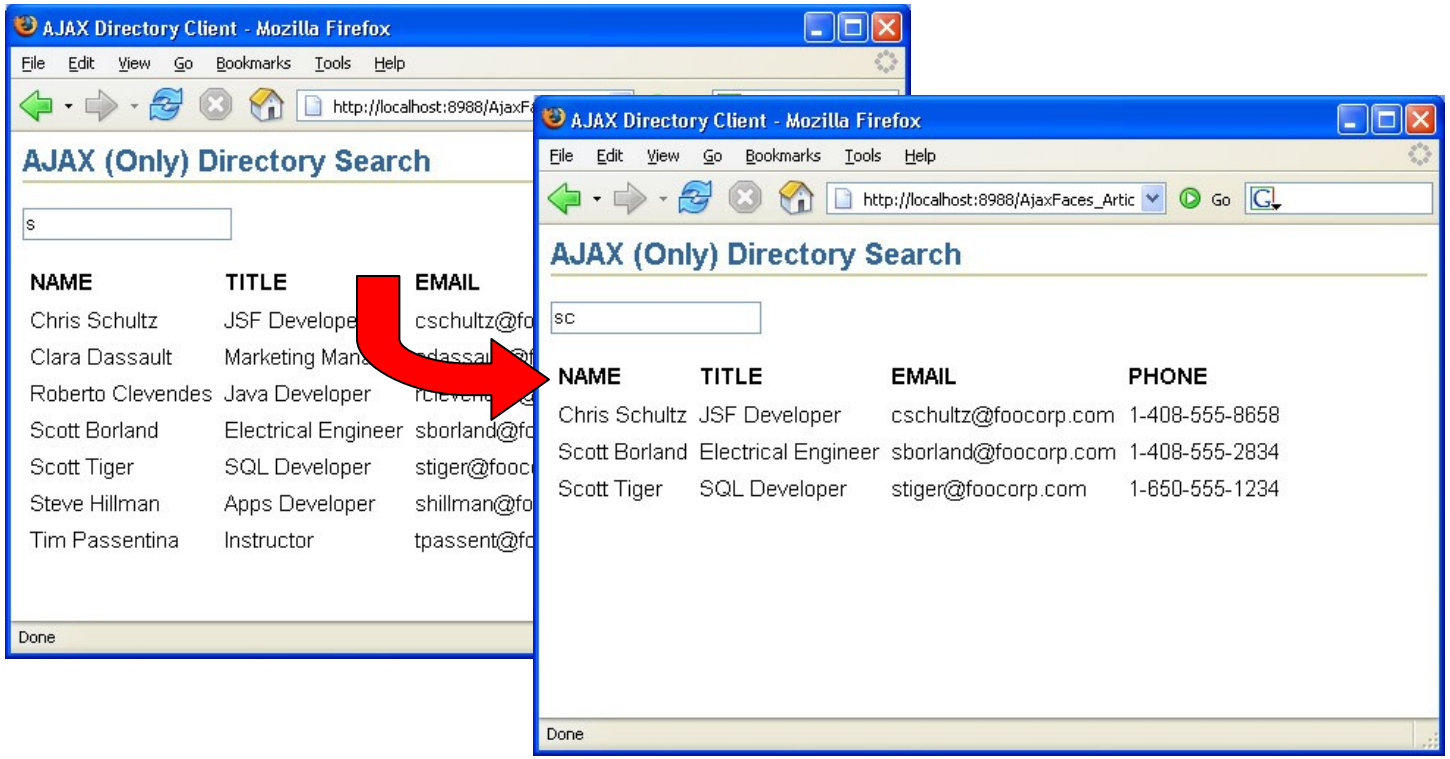
```
if (window.XMLHttpRequest) {
   req = new XMLHttpRequest();
}
else if (window.ActiveXObject) {
   req = new
ActiveXObject("Microsoft.XMLHTTP");
}

// To initiate request use

req.open("GET", url, true);
```

Using XMLHttpRequest

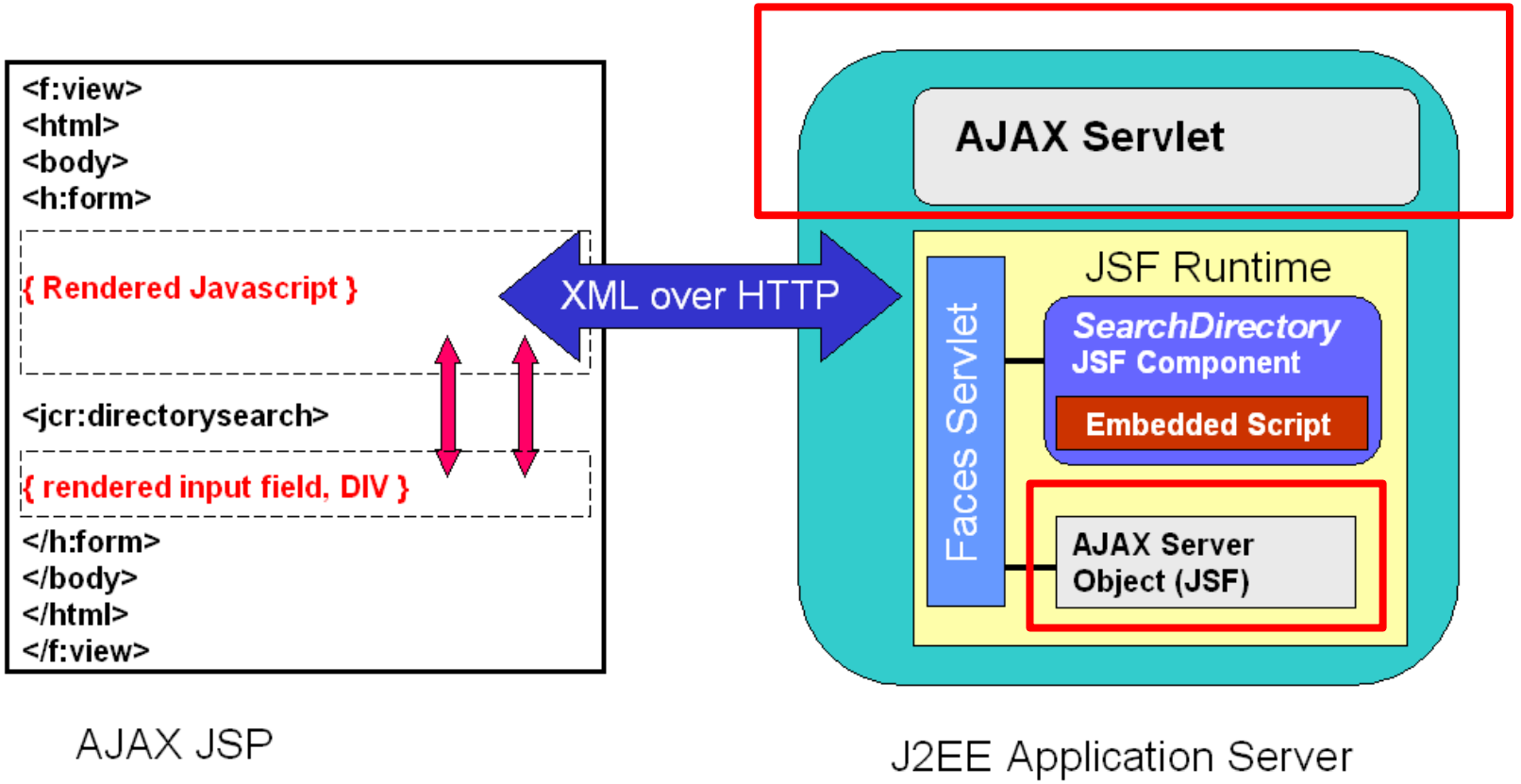
```
req.onreadystatechange = processXMLResponse;

// The function processXMLResponse( ), which processes the
// XML response, is invoked when the request is fulfilled.
// A callback function can also declared inline in the
// onreadystatechange statement:

req.onreadystatechange = processXMLResponse()
{
 // process request
};
```

DirectorySearch an AJAX JavaServer Faces Based Component

- Allows user to lookup employee directory information simply by keying in characters



DirectorySearch JavaServer Faces Based Component Architecture



DEMO

Building an AJAX directory search component

Strategies for Rendering JavaScript Code

- Can render static JavaScript based libraries separately
  - Can even provide tag/component for this
  - Slider uses this approach
- Component can render all JavaScript code necessary
  - Can check to see if JavaScript based library already rendered on client to prevent duplicates
- Can use PhaseListener, or Renderer code

Vendor Supplied Rich Client Components

DEMO

Oracle's ADF Faces Rich Client Components

Summary

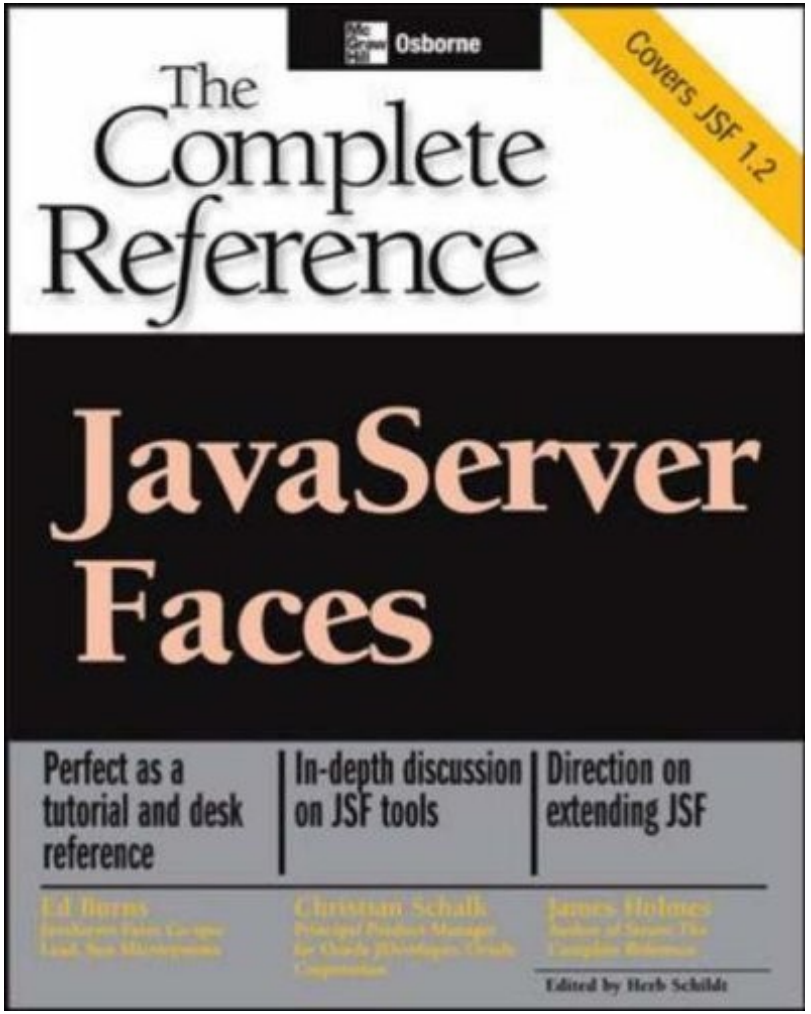
- JavaServer Faces Based Component Development is slightly complex at first, but soon reveals its powerful flexibility
- AJAX/Rich client components can be attained by rendering JavaScript code although there are different strategies to rendering JavaScript code
- Be creative!
  Use JavaServer Faces to draw new developers to Java world.

More Info

- http://jroller.com/page/cschalk
- http://jsfcentral.com
- http://otn.oracle.com/jsf
- http://otn.oracle.com/ajax

More Info

- Authors:
  Chris Schalk
  Ed Burns
- Tech Editor:
  Adam Winer
- Available summer 2006!



Q&A

Chris Schalk