Simplifying JavaServer™ Faces Component Development

Kito Mann

Author of *JavaServer Faces in Action*
Virtua, Inc.
www.virtua.com

Session TS-6178

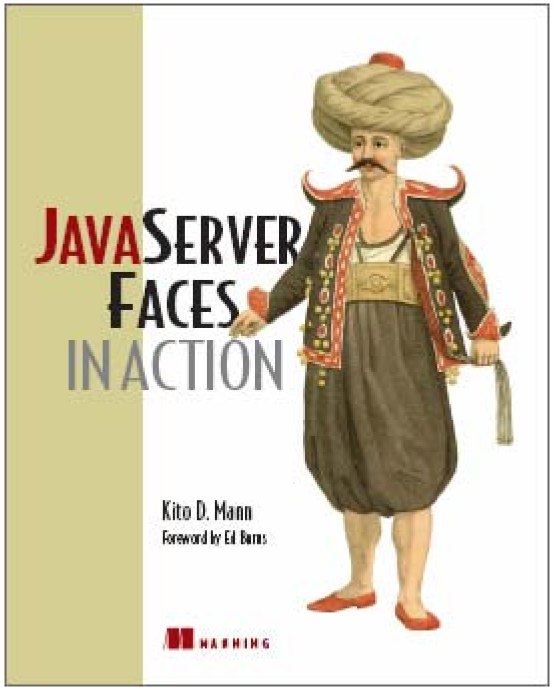
Simplifying JavaServer™ Faces Component Development

Techniques for rapidly developing JavaServer Faces components.

Writing JavaServer Faces components doesn't have to be a pain.

About Kito Mann

- Author, *JavaServer Faces in Action*
- Independent trainer, consultant, architect, mentor
- Internationally-recognized speaker
  - JavaOneSM Conference, JavaZone, TSS Symposium, Javapolis, NFJS, AJAX World, etc.
- Founder, JSF Central
  - http://www.jsfcentral.com
- Java Community ProcessSM (JCPSM) Member
  - JavaServer Faces 1.2, JavaServer Pages™ (JSP™) 2.1, Design-Time API for JavaBeans™ Architecture, Design-Time Metadata for JavaServer Faces Components, WebBeans, etc.
- Experience with Java™ Platform since its release in 1995, web development since 1993



Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

Intuition: a more optimal solution

Summary

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

Intuition: a more optimal solution

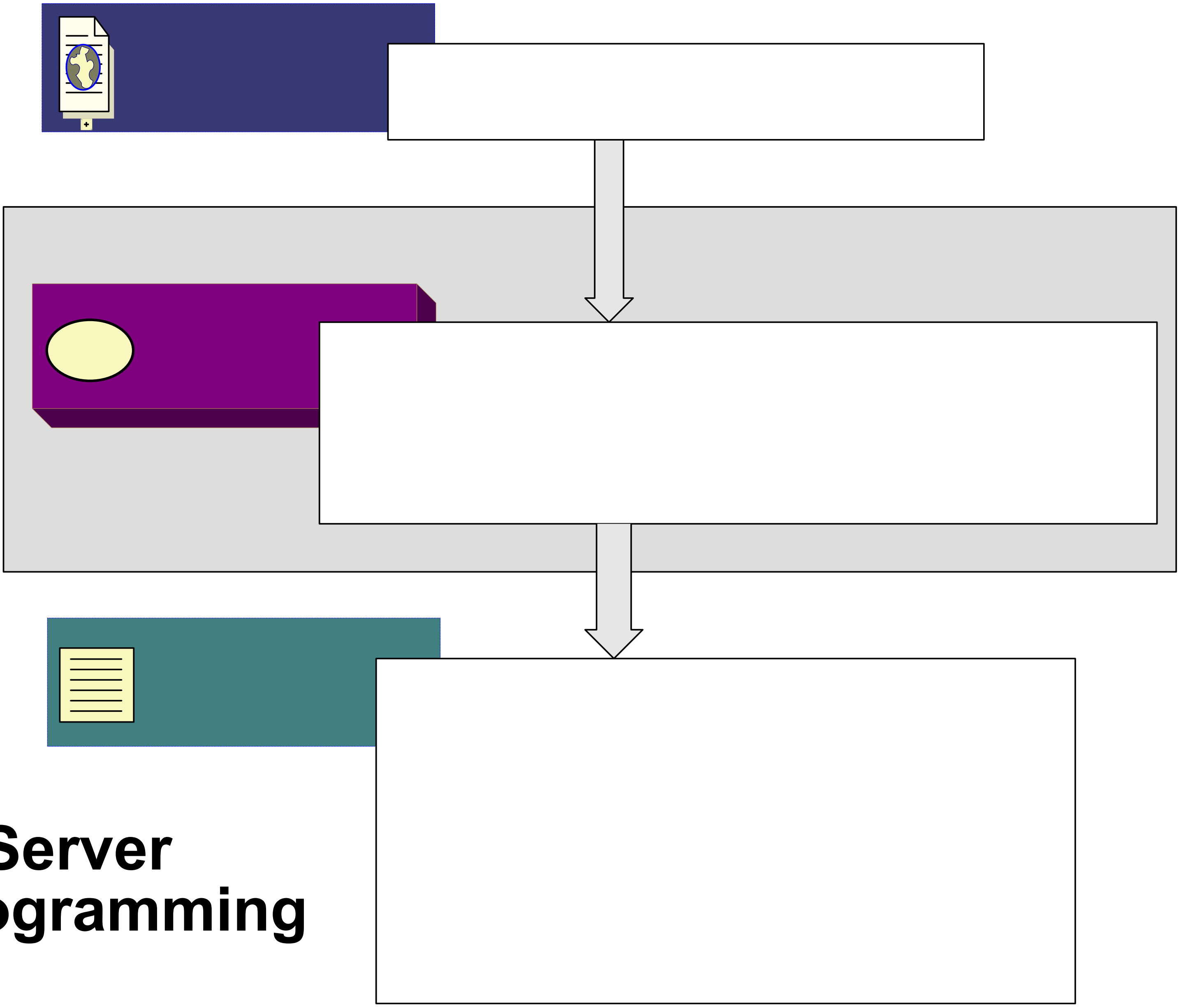
Summary

JavaServer Faces Components Overview

- Standard web user interface (UI) framework for Java technology
  - JavaServer Faces 1.0: Standardized through JCP Program in 2004 (JSR 127)
  - JavaServer Faces 1.2: Standardized through JCP Program in 2006 (JSR 252)
    - Part of Java Platform, Enterprise Edition (Java EE Platform) 5.0
- Specification consists of
  - Server side UI component and event model
  - Set of basic UI components
  - Basic MVC-style application infrastructure

JavaServer Faces Components Overview

- Can automatically synchronize UI components with application objects
- Basic Dependency Injection container
- Extensive tool support
  - Sun, Oracle, IBM, BEA, Exadel, Borland, JetBrains, Genuitec, others
- Enables RAD-style approach to Java platform web development
- Built on top of Servlet API
- Works with JSP software, but does not require it



The JavaServer Faces Programming Model

Key Feature: UI Components

- Standard UI component model enables a third-party component marketplace
  - Grids, Trees, Menus, Sliders, Panels, Charts, Popup Windows, Calendars, etc.
  - Open source and commercial vendors
  - Often have integrated AJAX support
- Components encapsulate complicated UI behavior
  - Saves development time

Ecosystem Players

- Apache Shale
- Apache Shale Clay
- Facelets
- JSFTemplating
- Weblets

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

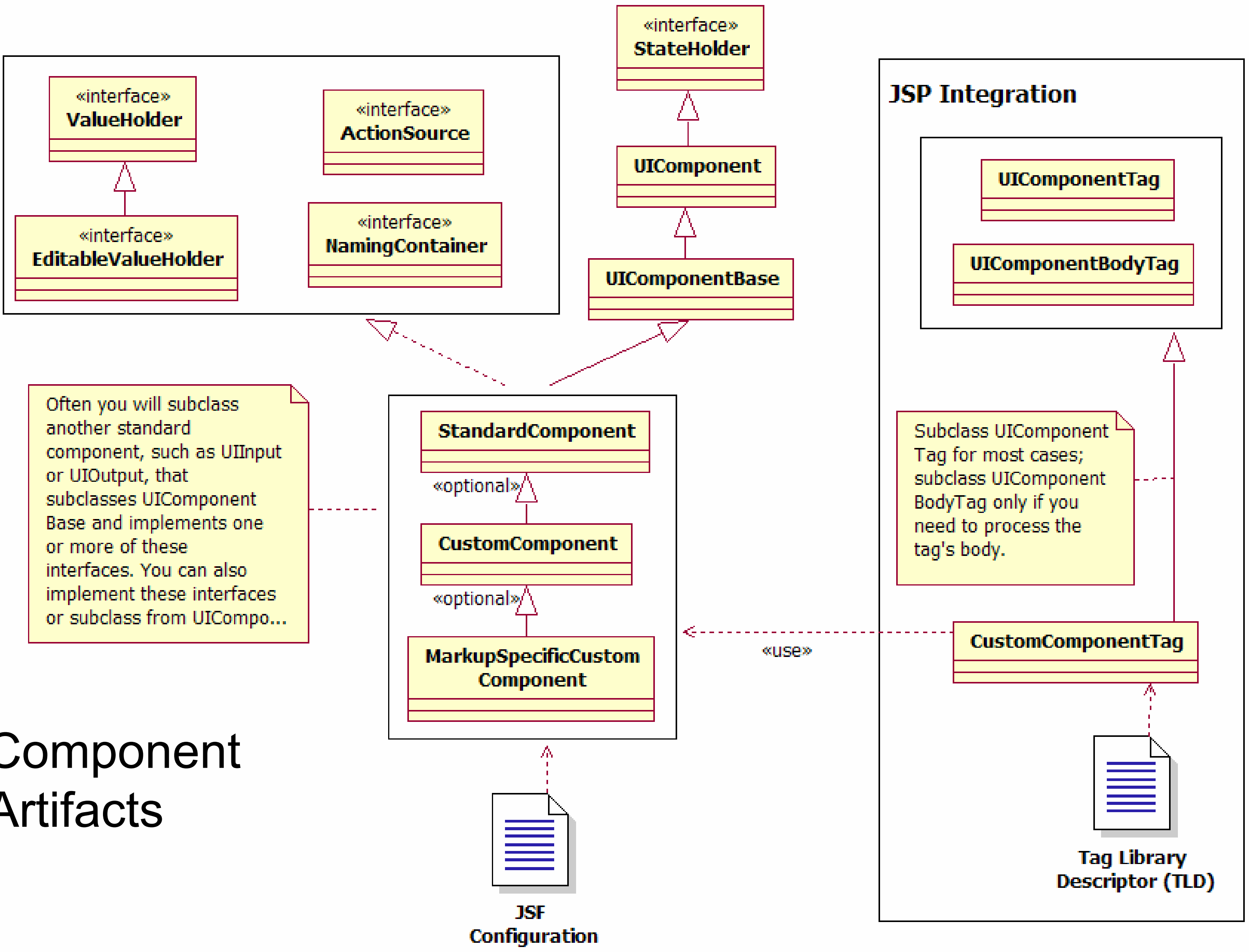
Intuition: a more optimal solution

Summary

UI Components the Standard Way

- Several artifacts
  - UI component class
  - Renderer class (optional)
  - Component registration
    - JavaServer Faces technology configuration entry
    - JSP software tag class
    - JSP software tag library entry

Component Artifacts



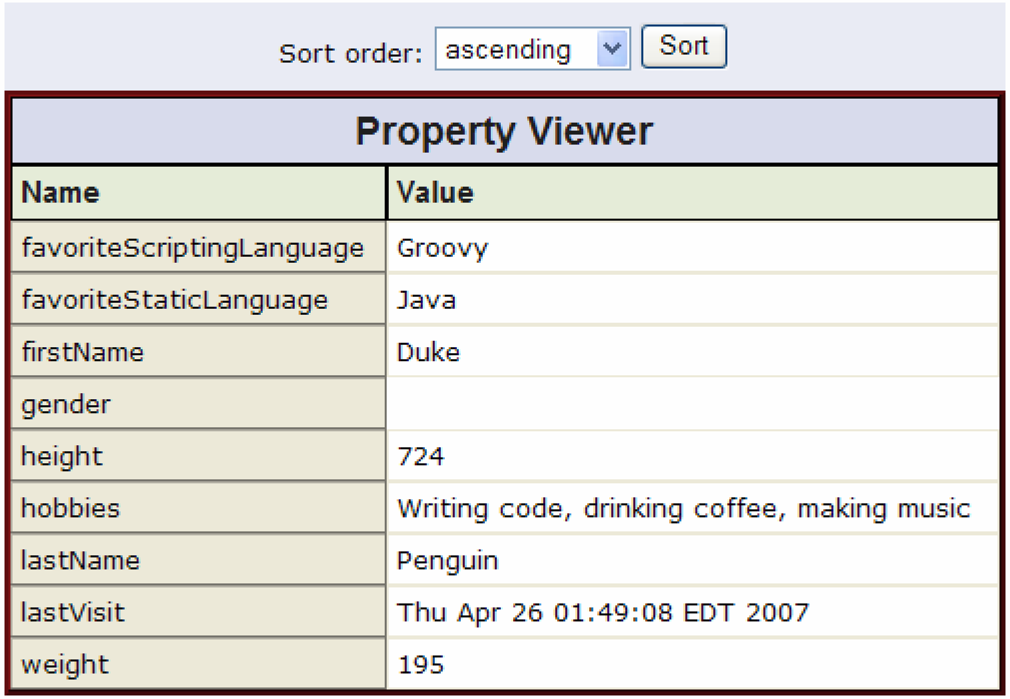
Often you will subclass another standard component, such as UIInput or UIOutput, that subclasses UIComponent Base and implements one or more of these interfaces. You can also implement these interfaces or subclass from UICompo...

Subclass UIComponent Tag for most cases; subclass UIComponent BodyTag only if you need to process the tag's body.

UI Components the Standard Way

Property Viewer example

```
<link type="text/css" rel="stylesheet" href="UIPropertyViewer.css" />
...
<i:propertyViewer value="#{demoBean}"  excludedProperties="class, weight"
    headerText="Property Viewer"
    style="border: rgb(128,19,21) outset 3px;" />
```



Sort order: ascending Sort

Property Viewer	
Name	**Value**
favoriteScriptingLanguage	Groovy
favoriteStaticLanguage	Java
firstName	Duke
gender	
height	724
hobbies	Writing code, drinking coffee, making music
lastName	Penguin
lastVisit	Thu Apr 26 01:49:08 EDT 2007
weight	195

UI Components the Standard Way: Declaring Properties

UIPropertyViewer.java

```
public String getStyle()
{
    if (this.style != null) {
      return this.style;
    }
    ValueExpression ve = getValueExpression("style");
    if (ve != null)
    {
        try {
             return ((String) ve.getValue(getFacesContext().getELContext()));
        }
        catch (ELException e) {
             throw new FacesException(e);
        }
    }
    return null;
}
public void setStyle(String style)
{
    this.style = style;
}
```

- Public properties must integrate with EL

UI Components the Standard Way: Encoding

UIPropertyViewer.java

```
@Override
 public void encodeBegin(FacesContext facesContext) throws IOException
 {
      if (isRendered())
      {
          if (!childComponentsConfigured)
          {
             configureChildComponents();
          }
         String selectId = getClientId(facesContext)
               + NamingContainer.SEPARATOR_CHAR + "sortSelect";

         ResponseWriter writer = facesContext.getResponseWriter();
         writer.startElement("span", this);
         writer.writeAttribute("class", "pv-sort", null);

         writer.startElement("label", this);
         writer.writeAttribute("for", selectId, null);
         writer.write("Sort order: ");
         writer.endElement("label");
```

UI Components the Standard Way: Encoding

UIPropertyViewer.java

```
        writer.startElement("select", this);
        writer.writeAttribute("id", selectId, null);
        writer.writeAttribute("name", selectId, null);
        for (SortType sortOrder : SortType.values())
        {
            writer.startElement("option", this);
            writer.writeAttribute("value", sortOrder, null);
            if (getSortOrder() != null && getSortOrder().equals(sortOrder))
            {
                writer.writeAttribute("selected", null, null);
            }
            writer.write(sortOrder.toString().toLowerCase());
            writer.endElement("option");
        }
        writer.endElement("select");
      ...
  }
```

UI Components the Standard Way: Configuring Child Components

UIPropertyViewer.java

```
protected void configureChildComponents()
{
    FacesContext facesContext = getFacesContext();
    ELContext elContext = facesContext.getELContext();
    Application application = facesContext.getApplication();
    ExpressionFactory expressionFactory =  application.getExpressionFactory();

    // Setup data table
    HtmlDataTable dataTable = (HtmlDataTable) application
                      .createComponent(HtmlDataTable.COMPONENT_TYPE);
    dataTable.setCellspacing("0");
    dataTable.setCellpadding("0");
    dataTable.setHeaderClass("pv-header");
    dataTable.setRowClasses("pv-row");
    dataTable.setColumnClasses("pv-column-odd,pv-column-even");
    setValueExpressionProperty(dataTable, "styleClass", "styleClass",
        styleClass);
    setValueExpressionProperty(dataTable, "style", "style", style);
    setValueExpressionProperty(dataTable, "value", "value", getProperties());
    dataTable.setVar("property");
    getChildren().add(dataTable);
```

UI Components the Standard Way: Configuring Child Components

UIPropertyViewer.java

```
    // Setup data table header
    HtmlOutputText header = (HtmlOutputText) application
    .createComponent(HtmlOutputText.COMPONENT_TYPE);
    setValueExpressionProperty(header, "value", "headerText", headerText);
    dataTable.getFacets().put("header", header);

    // Setup first column
    HtmlColumn column =
         (HtmlColumn) application.createComponent(HtmlColumn.COMPONENT_TYPE);
    column.setHeaderClass("pv-column-header");
    HtmlOutputText columnText =
         (HtmlOutputText) application.createComponent(HtmlOutputText.COMPONENT_TYPE);
    columnText.setValueExpression("value", expressionFactory
         .createValueExpression(elContext, "#{property.name}", String.class));
    column.getChildren().add(columnText);
    dataTable.getChildren().add(column);
  ...
}
```

Writing a JavaServer Faces Component with Intuition: Decoding

UIPropertyViewer.java

```
@Override
public void decode(FacesContext facesContext)
{
     if (isRendered())
     {
          Map parameterMap =
                 facesContext.getExternalContext().getRequestParameterMap();
                 String submittedSortOrder =
                       (String) parameterMap.get(getClientId(facesContext) +
                        NamingContainer.SEPARATOR_CHAR + "sortSelect");
          if (submittedSortOrder != null)
          {
                 setSortOrder(SortType.valueOf(submittedSortOrder));
          }
     }
}
```

UI Components the Standard Way: State Saving

UIPropertyViewer.java

```
// StateHolder methods

    public Object saveState(FacesContext context)
    {
         Object[] values = new Object[8];
         values[0] = super.saveState(context);
         values[1] = value;
         values[2] = excludedProperties;
         values[3] = headerText;
         values[4] = styleClass;
         values[5] = style;
         values[6] = childComponentsConfigured;
         values[7] = sortOrder;

         return values;
    }
```

UI Components the Standard Way: State Saving

UIPropertyViewer.java

```
public void restoreState(FacesContext context, Object state)
{
     values = (Object[]) state;
     super.restoreState(context, values[0]);
     value = values[1];
     excludedProperties = (String) values[2];
     headerText = (String) values[3];
     styleClass = (String) values[4];
     style = (String) values[5];
     childComponentsConfigured = (Boolean) values[6];
     sortOrder = (SortType) values[7];
}
```

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

Intuition: a more optimal solution

Summary

Simplifying Component Registration

- Two primary requirements
  - Registration with JavaServer Faces components
    - faces-config.xml
  - Registration with view technology
    - JSP technology, Facelets, Clay

Registration with JavaServer Faces Components

- Default: XML configuration
- Improvement: Annotations
  - Apache Shale
    - @FacesComponent
    - @FacesRenderer
    - Also has annotations for managed beans, converters, and validators

Registration with JavaServer Faces Components: Standard

WEB-INF/faces-config.xml

```
<faces-config>
    <component>
        <component-type>
            intuition.demo.UIPropertyViewer
        </component-type>
        <component-class>
            com.virtua.javaone.demo.components.UIPropertyViewer
        </component-class>
     </component>
</faces-config>
```

Registration with JavaServer Components: Shale Annotations

```
...
import org.apache.shale.tiger.register.FacesComponent;
...
@FacesComponent("intuition.demo.UIPropertyViewer")
public class UIPropertyViewer extends UIFaceletComponentBase implements
StateHolder
{
    ...
}
```

Registration with View Technology

- Default: JSP software Tag Handler and TLD entry
- Alternative: Single tag entry
  - Facelets tag library entry
- Alternative: No tag entry
  - Shale Clay

Registration with View Technology: JSP Software 2.1

PropertyViewerTag.java

```
package com.virtua.javaone.demo.components;
...
public class PropertyViewerTag extends UIComponentELTag
{
    private ValueExpression value;
    private ValueExpression styleClass;
    private ValueExpression style;
    private ValueExpression excludedProperties;
    private ValueExpression headerText;

    @Override
    public String getComponentType()
    {
         return "intuition.demo.UIPropertyViewer";
    }
```

Registration with View Technology: JSP Software 2.1

PropertyViewerTag.java

```
@Override
protected void setProperties(UIComponent component)
{
     super.setProperties(component);

     UIPropertyViewer viewer =
          (UIPropertyViewer) component;
     setProperty(viewer, “value", value);
     setProperty(viewer, "styleClass", styleClass);
     setProperty(viewer, "style", style);
     setProperty(viewer, "excludedProperties", excludedProperties);
     setProperty(viewer, "headerText", headerText);
}
```

Registration with View Technology: JSP Software 2.1

PropertyViewerTag.java

```
protected void setProperty(UIPropertyViewer viewer, String propertyName,
    ValueExpression expression)
{
    if (expression != null)
    {
        if (expression.isLiteralText())
        {
            viewer.setValueExpression(propertyName, expression);
        } else
        {
            viewer.getAttributes().put(propertyName,
            expression.getExpressionString());
        }
    }
}
```

Registration with View Technology: JSP Software 2.1

PropertyViewerTag.java

```
...
    public ValueExpression getValue()
    {
        return value;
    }
    public void setValue(ValueExpression value)
    {
        this.value = value;
    }
...
    @Override
    public void release()
    {
        super.release();
        value = null;
        styleClass = null;
        style = null;
        excludedProperties = null;
        headerText = null;
    }
}
```

- One property for each exposed attribute

Registration with View Technology: JSP Software 2.1

WEB-INF/intuition.tld

```
<taglib>
    <tlib-version>1.2</tlib-version>
    <short-name>i</short-name>
    <uri>com.virtua.intuition</uri>
    <tag>
        <name>propertyViewer</name>
        <tag-class>
            com.virtua.javaone.demo.components.PropertyViewerTag
        </tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>value</name>
            <required>no</required>
            <rtexprvalue>false</rtexprvalue>
            <deferred-value>
                <type>java.lang.Object</type>
            </deferred-value>
        </attribute>
        ...
    </tag>
</taglib>
```

- One attribute entry for each tag class property

Registration with View Technology: Facelets

WEB-INF/intuition.taglib.xml

```
<facelet-taglib>
    <namespace>com.virtua.intuition</namespace>
    <component>
        <tag-name>propertyViewer</tag-name>
        <source>UIPropertyViewer.xhtml</source>
    </tag>
</facelet-taglib>
```

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templates

Resolving resources

Intuition: a more optimal solution

Summary

Using Templates

- Templates simplify output of mark-up
- Default: Not available
- Alternative: Define subtrees that can be reused in different views
  - JSP software tag files, Facelets, Clay, JSFTemplating
  - Caveat: Not full-fledged components

Using JSP Software 2.1 Tag Files

WEB-INF/intuition.tld

```
<taglib>
    <tlib-version>1.0</tlib-version>
    <short-name>fn</short-name>
    <uri>com.virtua.intuition</uri>
    <function>
        <name>getProperties</name>
        <function-class>
            com.virtua.javaone.demo.PropertyWrapper
        </function-class>
        <function-signature>
          java.util.List getProperties(java.lang.Object,java.lang.String[])
        </function-signature>
    </function>
</taglib>
```

- Function (or backing bean) required for custom processing

Using JSP Software 2.1 Tag Files

WEB-INF/tags/propertyViewer.tag

```
<%@tag %>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@attribute name="value" deferredValue="true" %>
<%@attribute name="headerText" deferredValue="true" %>
<%@attribute name="excludedProperties" deferredValue="true" %>
<%@attribute name="style" deferredValue="true" %>
<%@attribute name="styleClass" deferredValue="true" %>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@taglib prefix="ifn" uri="com.virtua.intuition"%>

<h:dataTable var="property"
    value="#{ifn:getProperties(value, fn:split(excludedProperties, ','))}"
    styleClass="#{(styleClass == null) ? 'property-viewer' : styleClass}"
    style="#{style}"
    cellpadding="0" cellspacing="0"headerClass="pv-header"
    rowClasses="pv-row" columnClasses="pv-column-odd,pv-column-even">
```

Using JSP Software 2.1 Tag Files

WEB-INF/tags/propertyViewer.tag

```
    <f:facet name="header">
        <h:outputText value="#{headerText}"/>
    </f:facet>

    <h:column headerClass="pv-column-header">
        <f:facet name="header">
            <h:outputText value="Name"/>
        </f:facet>
        <h:outputText value="#{property.name}"/>
    </h:column>

    <h:column headerClass="pv-column-header">
        <f:facet name="header">
            <h:outputText value="Value"/>
        </f:facet>
        <h:outputText value="#{property.value}"/>
    </h:column>

</h:dataTable>
```

DEMO

JSP Software 2.1 Tag Files

Benefits of JSP Software 2.1

- Standard
- Better integration with JavaServer Faces technology than JSP software 2.0
- Changes reflected immediately
  - No need to restart the application
- Unified expression language
- Tag files
  - Can specify attribute names and types
  - Don’t require any TLD entries

Using Facelet Tag Files

WEB-INF/intuition.taglib.xhtml

```
<facelet-taglib>
    <namespace>com.virtua.intuition</namespace>
    <tag>
        <tag-name>propertyViewer</tag-name>
        <source>UIPropertyViewer.xhtml</source>
    </tag>
    <function>
        <function-name>getProperties
        </function-name>
        <function-class>
            com.virtua.javaone.demo.PropertyWrapper
        </function-class>
        <function-signature>
            java.util.List getProperties(java.lang.Object,java.lang.String[])
        </function-signature>
    </function>
</facelet-taglib>
```

- Must register tag file
- Function (or backing bean) required for custom processing

Using Facelet Tag Files

WEB-INF/UIPropertyViewer.xhtml

```
<span xmlns=http://www.w3.org/1999/xhtml xmlns:h="http://java.sun.com/jsf/html"
xmlns:f=http://java.sun.com/jsf/core xmlns:fn=http://java.sun.com/jsp/jstl/functions
xmlns:i="com.virtua.intuition">

<h:dataTable var="property"
    value="#{i:getProperties(value, fn:split(excludedProperties, ','))}"
    styleClass="#{(styleClass == null) ? 'property-viewer' : styleClass}"
    style="#{style}" cellpadding="0" cellspacing="0"
    headerClass="pv-header" rowClasses="pv-row"
    columnClasses="pv-column-odd,pv-column-even">

    <f:facet name="header">
        #{(headerText == null) ? 'Property Viewer' : headerText}
    </f:facet>

    <h:column headerClass="pv-column-header">
        <f:facet name="header">Name</f:facet>
        #{property.name}
    </h:column>
```

Using Facelet Tag Files

WEB-INF/UIPropertyViewer.xhtml

```
    <h:column headerClass="pv-column-header">
        <f:facet name="header">Value</f:facet>
        #{property.value}
    </h:column>

    </h:dataTable>
</span>
```

DEMO

Facelet Tag Files

Benefits of Facelets

- Built specifically for JavaServer Faces components
- Will heavily influence JavaServer Faces platform 2.0
- Changes reflected immediately
    - No need to restart the application
- Excellent error reporting
- You can use deferred expressions in template text

Agenda

JavaServer Faces and UI components
UI components the standard way
Simplifying component registration
Using templating
Resolving resources
Intuition: a more optimal solution
Summary

Resolving Resources

- Load resources (images, stylesheets, scripts) from classpath, Java Archive (JAR), or WEB-INF
- Default: not available
  - Should be implemented at Servlet level
- Alternative: resource serving extension
  - Weblets, Shale

Resolving Resources with Shale Remoting

WEB-INF/web.xml

```
<web-app>
...
    <context-param>
        <param-name>
            org.apache.shale.remoting.CLASS_RESOURCES
        </param-name>
        <param-value>
            /static/*:org.apache.shale.remoting.impl.ClassResourceProcessor
        </param-value>
    </context-param>
...
</web-app>
```

- Load resources from classpath:

```
<link type="text/css" rel="stylesheet”  href=“/static/UIPropertyViewer.css.jsf"/>
```

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

Intuition: a more optimal solution

Summary

Limitations of Existing Alternatives

- Component registration
  - Facelets still requires a tag library entry
  - Shale annotations do not address state saving
- Using tag files
  - Compositions are not full-fledged JavaServer Faces components
    - Can’t maintain state
    - Must encapsulate functionality in functions or backing beans
- Resolving resources
  - Facelets and other resources are not loaded from classpath (by default)

Benefits of Stateful Component Classes

- Have behaviour associated with the lifetime of the view
  - Backing beans cannot (by default) be associated with a specific template
- Can save state in-between requests
  - Participate in JavaServer Faces components' state-saving mechanism
- Can be easily manipulated in Java code
- Better tool integration

Intuition: A More Optimal Solution

- Component registration
  - Annotation support
  - No tag library entries
- Using templating
  - Components require
    - Standard Java component class
    - Faclets for defining output and/or child components
- Resolving resources
  - Facelet component templates loaded from classpath
  - Same name as Java class file
-

Intuition Dependencies

- Facelets
  - Templating
  - Tag integration
- Shale
  - Annotations
  - Remoting
- Some glue code

Writing a JavaServer Faces Component with Intuition: Registration and State Saving

UIPropertyViewer.java

```
package com.virtua.intuition.demo.components;
...
@FacesComponent("intuition.demo.UIPropertyViewer")
public class UIPropertyViewer extends UIFaceletComponentBase implements
StateHolder
{
    ...

    @FacesStatefulField private SortType sortOrder;
    @FacesStatefulField private Object value;
    @FacesStatefulField private String styleClass;
    @FacesStatefulField private String style;
    @FacesStatefulField private String excludedProperties;
    @FacesStatefulField private String headerText;

    ...
```

Writing a JavaServer Faces Component with Intuition

UIPropertyViewer.java

- Helper method for supporting expressions

```
...
    public Object getValue()
    {
         return getValueExpressionProperty("value", value);
    }

    public void setValue(Object value)
    {
         this.value = value;
    }
...
```

Writing a JavaServer Faces Component with Intuition: Encoding

- Expose attribute for template

```
@Override
public void encodeAll(FacesContext context) throws IOException
{
    this.getAttributes().put("sortValues", SortType.values());
    super.encodeAll(context);
}
```

Writing a JavaServer Faces Component with Intuition: Encoding

com/virtua/intuition/demo/components/UIPropertyViewer.xhtml

```
<span xmlns=http://www.w3.org/1999/xhtml xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:fn="http://java.sun.com/jsp/jstl/functions"
xmlns:h=http://java.sun.com/jsf/html xmlns:f="http://java.sun.com/jsf/core">

<link type="text/css" rel="stylesheet"
    href="#{component.resourceURL}/#{component.name}.css.jsf"/>

<span class="pv-sort">
<label for="#{component.clientId}:sortSelect">Sort order: </label>
<select id="#{component.clientId}:sortSelect"
    name="#{component.clientId}:sortSelect">
    <ui:repeat value="#{component.attributes['sortValues']}" var="sortOrder">
        <h:panelGroup rendered="#{component.sortOrder == sortOrder}">
            <option value="#{sortOrder}" selected="true">
                #{fn:toLowerCase(sortOrder)}</option>
        </h:panelGroup>
        <h:panelGroup rendered="#{component.sortOrder != sortOrder}">
            <option value="#{sortOrder}">#{fn:toLowerCase(sortOrder)}</option>
        </h:panelGroup>
    </ui:repeat>
</select>
```

Writing a JavaServer Faces Component with Intuition: Encoding

com/virtua/intuition/demo/components/UIPropertyViewer.xhtml

```
    <input type="submit" value="Sort" class="pv-button"/>
</span>

<h:dataTable var="property" value="#{component.properties}"
    styleClass="#{component.styleClass}" style="#{component.style}"
    cellpadding="0" cellspacing="0"headerClass="pv-header"
    rowClasses="pv-row" columnClasses="pv-column-odd,pv-column-even">

    <f:facet name="header">
        #{component.headerText}</f:facet>
    <h:column headerClass="pv-column-header">
        <f:facet name="header">Name</f:facet>
        <span>#{property.name}</span>
    </h:column>
    <h:column headerClass="pv-column-header">
        <f:facet name="header">Value</f:facet>
        <span>#{property.value}</span>
    </h:column>
</h:dataTable>
</span>
```

Writing a JavaServer Faces Component with Intuition: Decoding

UIPropertyViewer.java

- Normal behavior

```
@Override
public void decode(FacesContext facesContext)
{
    if (isRendered())
    {
        Map parameterMap =
                facesContext.getExternalContext().getRequestParameterMap();
                String submittedSortOrder =
                     (String)  parameterMap.get(getClientId(facesContext) +
                      NamingContainer.SEPARATOR_CHAR + "sortSelect");
         if (submittedSortOrder != null)
        {
                setSortOrder(SortType.valueOf(submittedSortOrder));
        }
    }
}
```

Writing a JavaServer Faces Component with Intuition

demo.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core" xmlns:i="com.virtua.intuition">

<body>
<h:form>

<i:propertyViewer value="#{demoBean}"
    excludedProperties="class,weight"
    headerText="Intuition Property Viewer"
    style="border: rgb(128,19,21) outset 3px;" />

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

- Tag automatically generated based on class name

DEMO

Intuition

Agenda

JavaServer Faces and UI components

UI components the standard way

Simplifying component registration

Using templating

Resolving resources

Intuition: a more optimal solution

Summary

Summary

- Developing JavaServer Faces components is tedious
  - Component registration
    - JavaServer Faces component registration
    - JSP software integration
  - No template support for output
  - Resource resolution not integrated
- Alternatives
  - Component registration
    - Apache Shale annotations
    - Facelets

Summary

- Templating
  - JSP software 2.1 tag files
  - Facelet tag files
- Resource resolution
  - Apache Shale remoting
- Intuition: a more optimal solution
  - Uses Shale annotations and Facelets
  - Requires only a component class and a template

For More Information

- JavaServer Faces in Action, Kito D. Mann
  - http://www.manning.com/mann
- Official JavaServer Faces Technology Site
  - http://java.sun.com/javaee/javaserverfaces/
- JSFCentral
  - http://www.jsfcentral.com
- Facelets
  - http://facelets.dev.java.net
- Apache Shale
  - http://shale.apache.org

For More Information

- Sessions and BOFs
  - TS-7082—Building JavaServer Faces Applications with Spring and Hibernate
  - TS-4439—Minimalist Testing Techniques for Enterprise Java Technology-Based Applications
  - BOF-4400—Improve and Expand JavaServer Faces Technology with JBoss Seam
  - TS-4514—Three Approaches to Securing Your JavaServer Faces Technology/Spring/Hibernate Applications

Q&A

Kito Mann

Principal Consultant,
Virtua, Inc.

Author of *JavaServer Faces in Action*

www.virtua.com
www.jsfcentral.com

Virtua™

technical excellence. it's that simple.

Simplifying JavaServer™ Faces Component Development

Kito Mann

Author of *JavaServer Faces in Action*
Virtua, Inc.
www.virtua.com

Session TS-6178