Apache MyFaces
Open Source JavaServer Faces

Matthias Weßendorf
Martin Marinschek
Mario Ivankovits

ApacheCon Europe 06

Table of Content

- Introduction JSF basic
- Introduction to Apache MyFaces
- Some goodies of MyFaces
  - Tomahawk
  - Sandbox
- all „goodies“ are supported with Demos
- Conversation Tag
- Discussion (or Question & Answer)

What is JavaServer Faces? (1)

- Framework/Standard for Java-Web-Development
- With experience: much easier than plain JSP/Servlet
- POJO for Web Development (backing beans)
- Managed Bean Facility - IoC (setter injection)

What is JavaServer Faces? (2)

- provides UI components similar to Swing
- default RenderKit for HTML 4.0.1
  - other Renderkits possible: XHTML, WML, etc.

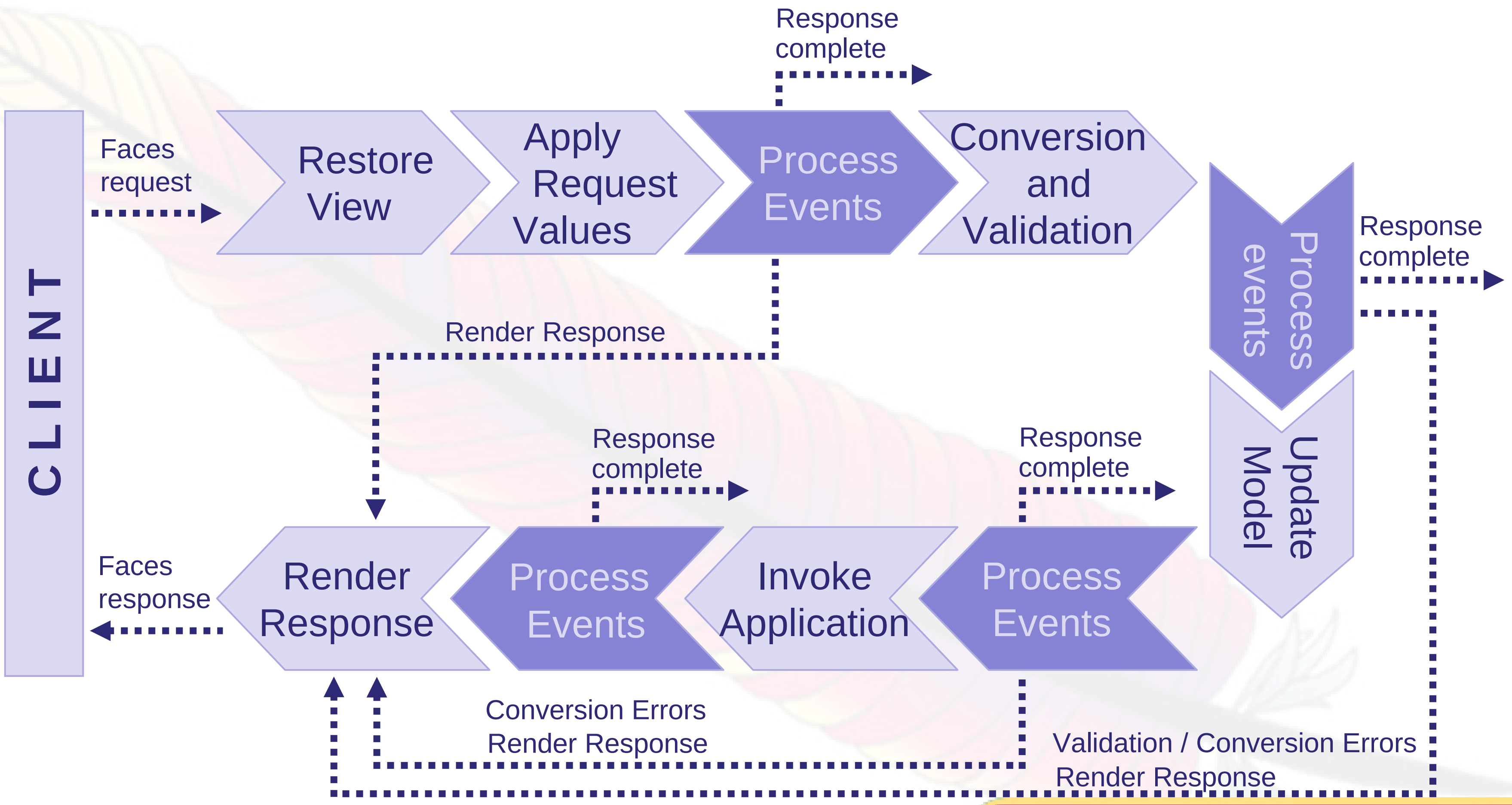
```
<h:inputText id="x" />
```

$\Rightarrow$

```
<input type="text" id="form:x"/>
```

- GUI Event Handling (JavaBean Standard)
- auto converter / custom converters
- standard validators / custom validators

JSF Lifecycle



JSF – Hello World (JSP file)

```
<h:form id="form">
  <h:panelGrid columns="2">
    <h:outputLabel for="input1">
      <h:outputText id="input1Label" value="first name"/>
    </h:outputLabel>
    <h:inputText id="input1" required="true"
      value="#{customer.firstname}"/>

    <h:outputLabel for="input2">
      <h:outputText id="input2Label" value="second name"/>
    </h:outputLabel>
    <h:inputText id="input2" value="#{customer.secondname}"
      required="true"/>

<h:commandButton value="send it!" action="#{customer.send}"/>

    <h:messages style="color:red" layout="table"/>
  </h:panelGrid>
</h:form
```

JSF – Hello World (JavaBean)

```
public class Customer {

private String firstname = null;
private String secondname = null;
//getter and setter

  public String send(){
   //back-end access (e.g. BusinessDelegate)
    return ("fine");
  }
}
```

JSF – Hello World (XML file)

```
<faces-config>
  <managed-bean>
 <managed-bean-name>customer</managed-bean-name>
 <managed-bean-class>foo.Customer</managed-bean-
   class>
 <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  <navigation-rule>
     <from-view-id>/form.jsp</from-view-id>
     <navigation-case>
       <from-outcome>fine</from-outcome>
       <to-view-id>/output.jsp</to-view-id>
     </navigation-case>
  </navigation-rule>
</faces-config>
```

Why JavaServer Faces?

- Industrial Standard via JCP
  - JSR 127 (JSF 1.0 and JSF 1.1) 2004
  - JSR 252 (JSF 1.2) end of 2005 ?
    - JSF 1.2 works with JSP 2.1 and solves some issues
  - JSF 2.0 (AJAX, more UI components, ..)

    Perhaps 2006?
- Part of Java EE 5.0
- has big vendor support
  - IDE support (Sun, Eclipse, Oracle, ...)
  - 3rd party UI-components (Apache MyFaces, Oracle ADF Faces, Atanion Tobago)

JSF Implementations

- Sun (RI)
- Simplica (based upon Apache MyFaces)
- IBM
- Apache MyFaces

  - additionally, there are several 3rd party UI components that *should* run with *any* implementation.

Apache MyFaces

- Founded in 2002 by Manfred Geiler and Thomas Spiegl
  - sourceforge and LGPL based
- In July 2004 moving to Apache Software Foundation (Incubator)
- Since February 2005 TLP (myfaces.apache.org)
- 19 developers
- lot's of contributor (maybe you soon too ?)
- 1.1.x STABLE!
  - since 19th September 2005 Apache MyFaces is JSF spec compilant!

MyFaces provides:

- Implementation of JSF-API
  - `javax.faces.**` Classes
- Implementation of JSF Spec
  - `org.apache.myfaces.**` Classes
- Custom Components
  - Scrollable Table, Validators, Tree components ...
- Custom extensions
  - Built-in Tiles support, RenderKit for WML/WAP
- Support for Portlet Spec (JSR 168)
  - MyFaces apps runs in Pluto, JBoss Portal and some others.

How MyFaces is structured ?

- A core
  - contains the JSF-API and time runtime (aka IMPL)
- A “familiy” of components set (aka the MyFaces components)
  - Tomahawk (“stable” components)
  - Sandbox („experimental“ components)
  - TOBAGO (Layout oriented stuff)

MyFaces compatibility (tested)

- Java 1.4 and Java5
- Tomcat (4.1.x, 5.0.x and 5.5.x)
  - for Tomcat 5.5 you need to delete two jars (see doc for info)
- JBoss (3.2.x and 4.0.x)
- JRun4
- Bea Weblogic 8.1
- Jonas 3.3.6 w/ Tomcat
- Resin 2.1.x
- Jetty 4.2
- Websphere 5.1.2
- OC4J
- Orion (see wiki page for instructions)

MyFaces Internals I

- StartupServletContextListener
  - inits XML config processing
  - inits the „JSF Infrastructure“
  - must *not* be registered in **your** web.xml
    - only in case of some containers…
- ExtensionsFilter
  - used during upload (parses Multipart requests)
  - adds resources (images, js,…) that are needed by components (easier to reuse components)
  - good performance

MyFaces Internals II

- special Servlet Context parameter
  - ALLOW_JAVASCRIPT
  - AUTO_SCROLL
  - PRETTY_HTML
  - ...
- dummy form for commandLinks
  - no need to wrap your <h:commandLink/> components inside of UIForm (<h:form/>)

MyFaces in Action (Tomahawk)

- several custom components
- custom validator components
- custom extensions

Custom calendar component

- Renders as a form:

```
<x:inputCalendar ...
  value="#{travel.arrival}" />
```

- Renders as a popup:

```
<x:inputCalendar ...
   renderAsPopup="true"
   value="#{travel.depature}" />
```

- sample

Custom Upload Component

- Upload is **not** part of JSF spec (currently)
- uses Servlet Filter (MyFaces‘ Extension Filter)
- based upon Jakarta Commons' FileUpload
- special MyFaces interface:
  `org.apache.myfaces.custom.fileupload.`**`UploadedFile`**

```
<h:form enctype="multipart/form-data">
  <x:inputFileUpload
    value="#{backing.file}“
    required="true"/>
  …
</h:form>
```

- sample

Tree Component (Tree2)

- MyFaces provides two tree components
- define your data inside a backing bean
  - TreeNode (Interface)
  - TreeNodeBase (Implementation class)
- define your layout in a JSF page via facets
- Navigation via CommandLink component
- client and server toggle

Tree Component Java code

```
private TreeNode tree;
tree = new
  TreeNodeBase(„folder“,“navi“,true);

tree.getChildren().add(
new TreeNodeBase(„doc“,“entry“,false)
)
```

Tree Component JSP

```
<x:tree2 value=„#{bean.tree}“ clientSideToggle=„true“
   var=„node“ varNodeToggle=„t“ ...>
<f:facet name=„doc">

<h:panelGroup>
   <h:commandLink styleClass="document" action=„nav">
   <h:graphicImage value="images/document.png„
                            border="0"/>
   <h:outputText value="#{node.description}"/>
  <f:param name=„reqVal" value="#{node.identifier}"/>
  </h:commandLink>
</h:panelGroup>

</f:facet>
...
</x:tree2>
```

- sample

Tabbed Pane

- Tab control as known from classic GUIs
- Contains two custom JSF tags
  - `<x:panelTabbedPane/>`
  - `<x:panelTab/>`
- reuses standard UI components
  - for instance `<h:inputText/>`
- click on a tab ends up in dynamic html (no req)
- tab saves the state of the nested input fields

Tabbed Pane JSP code

```
<x:panelTabbedPane bgcolor=„#FFFFCC“>

 <x:panelTab id=„tab1“ label=„Main Menu“>
      <h:outputText .../>
     <h:inputText value=„#{bean.property}“/>
      ...
 </x:panelTab>
 <x:panelTab id=„tab2“ label=„second Menu“>
      ...
 </x:panelTab>
<h:commandButton value=„Submit it!“ />
</x:panelTabbedPane>
```

- sample

custom Table component

- MyFaces contains a custom table component
- extends UIData (standard component)
  - preserveDataModel
  - sortColumn
  - sortAscending
  - preserveSort
  - renderedIfEmpty
  - rowIndexVar

scrollable Table component

```
<x:dataTable id="data“ …>
...
</x:dataTable>

<x:dataScroller id="scroll_1“ for="data“ fastStep="10“
   pageCountVar="pageCount“ pageIndexVar="pageIndex“
   styleClass="scroller“ paginator="true“
   paginatorMaxPages="9“ paginatorTableClass="paginator“
   paginatorActiveColumnStyle="font-weight:bold;">

<f:facet name="first" >
<h:graphicImage url="images/arrow-first.gif" border="1" />
</f:facet>
...
</x:dataScroller>
```

- sample

sortable Table component

- needs MyFaces <x:dataTable/> attributes:
  - `sortColumn="#{sorter.sort}"`
  - `sortAscending="#{sorter.asc}"`
  - `preserveSort="true"`
- uses MyFaces <x:dataTable/> BackingBean needs method (`sort()`) that contains a `Comparator` impl.
- call `sort()` before returning the data model.
  - here: call inside of `getWorkers();`
- sample

Using *Legacy* JavaScript and CSS

- JSF Components using IDs:

```
<h:form id=„foo“>
<h:inputText id=„bar“ ... >
</h:form>
```

generates **foo:bar**

- document.getElementById();
- special forceId Attribute (JSF 1.2 contains a similar concept):

```
<h:form id=„foo“>
<x:inputText id=„bar“ forceId=„true“... >
</h:form>
```

generates **bar**

Custom Validators

- nest them inside Input Components

```
<h:inputText value=„...>
   <x:validateEmail/>
</h:inputText>
```

- ISBN (`<s:validateISBN/>`)
- CreditCard (`<x:validateCreditCard/>`)
- Regular Expression
  `<x:validateRegExpr pattern=„\d{5}“/>`
- Equal

```
<h:inputText id=„password1“ ... />
<h:inputText id=„password2“ ...>
   <x:validateEqual for=„password1“/>
</h:inputText>
```

UpdateActionListener I

```
JSP:
<h:dataTable var="emp" .... >
<h:commandLink id="editLink" action="details">
  <h:outputText value="#{msg.edit}"/>
  <f:param name="id" value="#{emp.id}"/>
</h:commandLink>

backing bean:
FacesContext context = FacesContext.getCurrentInstance();
Map map =
context.getExternalContext().getRequestParameterMap();

String employeeID = (String) map.get("id");

businessDelegateMethod(employeeID);
```

UpdateActionListener II

```
<h:dataTable var=”user” ...>

<h:commandLink action="#{userbacking.deleteUser}"
   value="Delete">

   <t:updateActionListener
      property="#{userbacking.currentUser}"
      value="#{user}" />
</h:commandLink>
```

JSF – composing pages

- Standard provides a plain subview „component“
  - `<jsp:include />` or `<c:import />`
- realizes the Composite View Pattern
- bound to file names (e.g. `footer.jsp`)
- good framework for composing pages
  - Tiles (used in Struts, Velocity or plain JSP)

MyFaces Tiles integration

- custom ViewHandler for Tiles
  - must be registed in `faces-config.xml`
  - needs tiles configuration location as ContextParameter (`web.xml`)
  - looks up `*.tiles` mappings in tiles definition file
  - page definitions are described in `tiles.xml`

MyFaces/Tiles - definitions

```
<tiles-definitions>
<definition name="layout.example“
   path="/template/template.jsp" >
  <put name="header" value="/common/header.jsp" />
  <put name="menu" value="/common/navigation.jsp" />
</definition>

<definition name="/page1.tiles"
   extends="layout.example" >
  <put name="body" value="/page1.jsp" />
</definition>

</tiles-definitions>
```

MyFaces/Tiles – master template

```
<table>
<tr><td>
<f:subview id="menu">
  <tiles:insert attribute="menu" flush="false"/>
</f:subview>
</td>

<td>
<f:subview id="body">
  <tiles:insert attribute="body" flush="false"/>
</f:subview>
</td>
</tr>
</table>
```

- sample

MyFaces‘ WML RenderKit

- supports basic JSF components to render in WAP devices
- supports WML and **not** XHTML MP (WAP2.0)
- add WML RenderKit to `faces-config.xml`
- uses XDoclet to generate components, tag classes and tld file
- contribution from Jiri Zaloudek

WML RenderKit - code

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">

<%@ page contentType="text/vnd.wap.wml" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"
   %>
<%@ taglib uri="http://myfaces.apache.org/wap"
   prefix="wap" %>

<wml> <card id="helloId" title="Hello WML World">
<p> <f:view>

<wap:form id="form">
<wap:outputText id="label" value="Your name"/>
<wap:inputText id="name" value="#{hello.yourname}" />
<wap:commandButton id="submit" action="#{hello.send}“
   value="submit it" />
</wap:form>
</f:view>
...
```

sample

MyFaces – Portlet support

- Built-in-support for JSR 168
- contribution by Stan Silvert (JBoss Group)
- what must a user do?
  - Make sure your JSF MyFaces application runs as a stand-alone servlet.
  - Remove any redirects from your faces-config.xml. Portlets can not handle these.
  - Create a Portlet WAR as per the instructions for your Portlet container. Make sure it contains everything that was included in step 1.
  - Update your portlet.xml

MyFaces – portlet.xml

```
<portletclass>
org.apache.myfaces.portlet.MyFacesGenericPortlet
</portlet-class>

<init-param>
 <name>default-view</name>
 <value>/some_view_id_from_faces-config</value>
</init-param>

<init-param>
 <name>default-view-selector</name>
  <value>com.foo.MyViewSelector</value>
</init-param>
```

MyFaces in Action II (Sandbox)

- „experimental component set of the MyFaces community, but…
  100% buzzword compliant :-D
- AJAX based components
- Scheduler component
- Formular extension
- URL Validator
  - and more …

Scheduler

- Renders a schedule component
- appointments and events in a day, workweek, week or month view
- Themes for Ximian's Evolution for instance
- Java-API:
  - ScheduleModel (Interface)
    - AbstractScheduleModel and SimpleScheduleModel
  - ScheduleEntry (Interface)
    - DefaultScheduleEntry

Scheduler – Java Code I

```
private ScheduleModel model = new
  SimpleScheduleModel();
private Integer mode = new
  Integer(ScheduleModel.DAY);

model.setMode(this.mode);
model.setSelectedDate(this.date);
    //java.util.Date object
```

Scheduler – Java code II

```
DefaultScheduleEntry entry = new
  DefaultScheduleEntry();
entry.setTitle(„TEST“);
entry.setStartTime(date);
entry.setEndTime(date2);
    //java.util.Date object

((SimpleScheduleModel)model).addEntry
  (entry);
```

Scheduler – JSP code

```
<f:view>
<h:form>
<s:schedule
  value="#{scheduler.model}"
  id="schedule1"
  theme="evolution" />
</h:form>
</f:view>
```

Visual Effect components

- based upon script.aculo.us
- easy usage in your application:

```
<s:effect id=”ef1” puff=”true”>
 <h:outputText value=“Hello Apache“/>
</s:effect>
<s:effect id=”ef2” fade=”true”>
 <t:outputText value=“Hello Apache“/>
</s:effect>
```

☐ sample

Long term visions for MyFaces

- Implementation of JSF 1.2
- Passing the TCK for JSF 1.2
- Growing our set of components (JavaScript, AJAX enabled components) and extensions

- Motivating **you** gals and guys to contribute!!!!

Conversation Tag

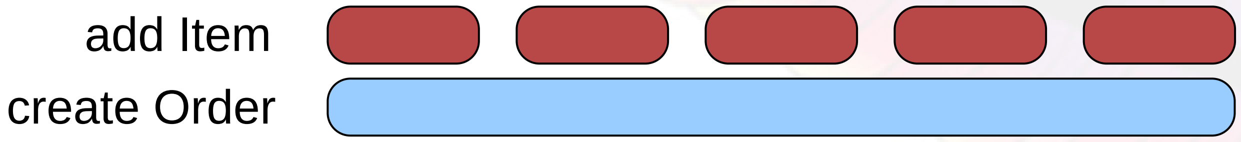
MyFaces

Conversation Tag
Build a dialog with JSF tags only.

Mario Ivankovits

Application Transaction

- The application transaction spans across multiple requests
- Extract conversation dependent state into its own bean
- Only store object identifiers outside a „persistent“ conversation
  - maybe use a nested conversation



Features

- Multiple window aware
  - parallel running conversationContexts
- Nested (Named) conversations
  - multiple conversations per conversation context
- Handle persistence (persistenceContext, e.g. Hibernate, EntityManager, ...)
- Conversation Timeout
- Works without serialization

s:startConversation

- This will start a conversation
  - per page (most likely)
  - on action
- You have to provide a conversation name
- Set the persistence flag to true if the conversation requires a persistenceContext

s:conversation

- Get a bean out of its current scope and put it into a named conversation scope
- Bean will be destroyed when conversation ends
- Keep track of conversation state with the ConversationListener interface

s:endConversation

- This will end a conversation
  - per page
  - on action (most likely)
    - on specific outcome
- Invokes a navigation on failure
  - clean restart
- Removes conversational beans

Additional Tags

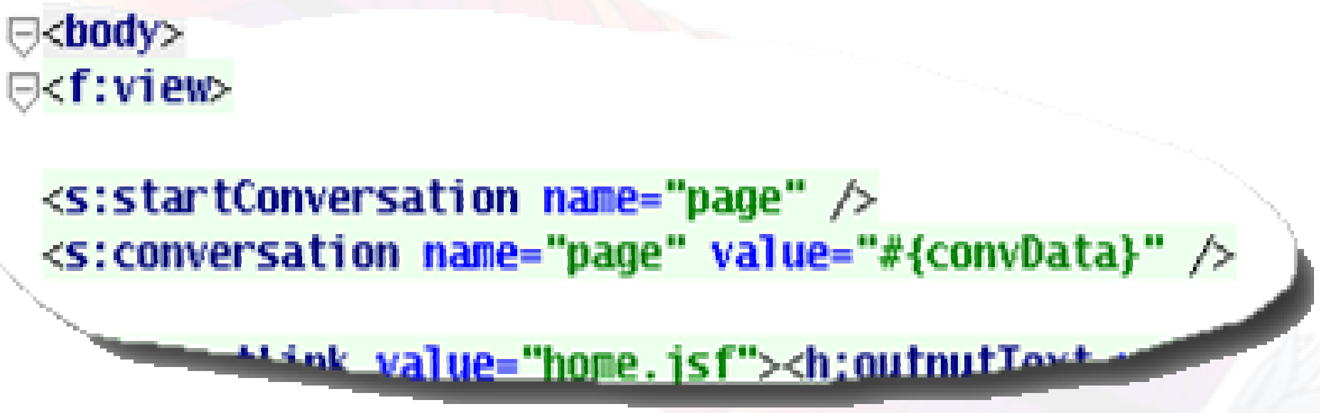
- s:separateConversationContext
  - links as child within this tag will start a new conversationContext
- s:ensureConversation
  - ensures a named conversation is active, else will redirect to the given view

Samples

- MyFaces tomahawk sandbox examples
- pageConversation.jsp
  - emulation of the „open session in view“ pattern
- wizard*.jsp
  - application transaction which spans multiple views

pageConversation.jsp

- ✓ Start a conversation and upgrade the bean „convData“ to the conversation scope …



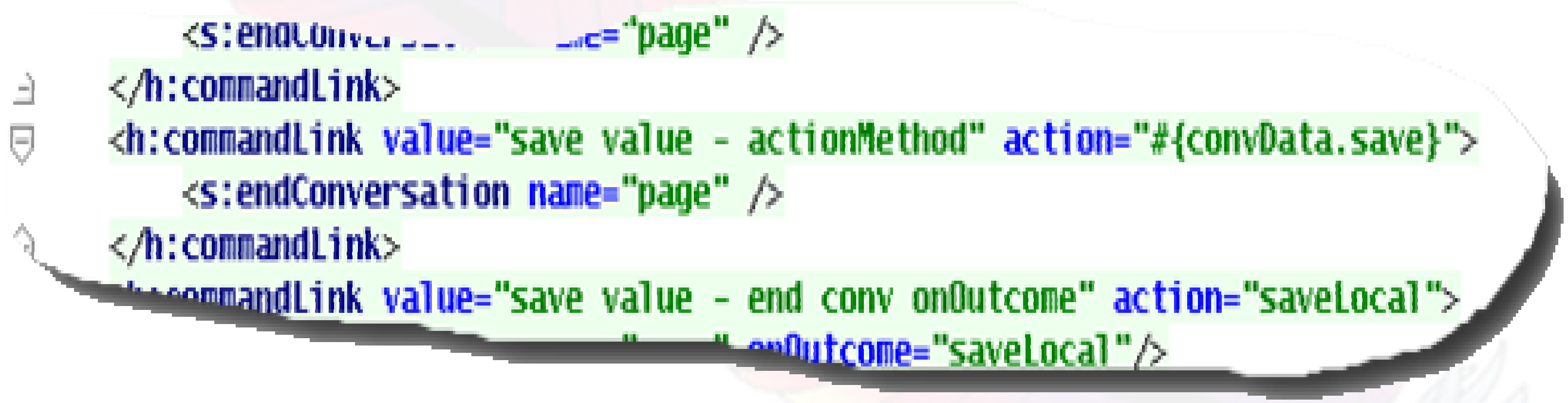
```
<body>
<f:view>

<s:startConversation name="page" />
<s:conversation name="page" value="#{convData}" />

    ...link value="home.jsf"><h:outputText ...
```

pageConversation.jsp

- … and the conversation end „on action“



```
    <s:endConv        e="page" />
</h:commandLink>
<h:commandLink value="save value - actionMethod" action="#{convData.save}">
    <s:endConversation name="page" />
</h:commandLink>
  commandLink value="save value - end conv onOutcome" action="saveLocal">
                      onOutcome="saveLocal"/>
```

Links and books

- JSF books
  - „JSF in Action“ by Kito D. Mann
  - „JavaServer Faces“ by Hans Bergsten
  - David Geary „Core JSF“
- JSF web resources
  - JSFCentral.com
  - jamesholmes.com/javaserverfaces
  - myfaces.apache.org
- Framework for JSF:
  - Struts Shale (completely based on JSF)
    - wiki.apache.org/struts/StrutsShale

Questions ?

- Answers!