JUDCon

JBoss Users & Developers Conference

Boston:2011

Realize the full potential of JSF with Seam Faces and CDI

Brian Leathem - Seam Faces Module Lead
http://twitter.com/brianleathem

What's this all about?

Seam 3

Seam Faces

JSF 2 / CDI integration is not as thorough is it could have been.
How does Seam Faces solve these problems?

How are these solutions implemented in Seam Faces?

JSF? CDI? What?

Show of hands for those in the audience who:

- are familiar with JSF?
- actively develop with JSF?
- JSF 2?
- are familiar with CDI?
- are familiar with Seam 2?

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

Seam - Innovation & Standardization

Seam 2 married JSF 1.x with EJB 3

Much of Seam 2 was standardized in JSF 2.0 and CDI

JSF is Good!

Part of the Java EE standard

Many groups working to make things easier for developers.

These improvements are turned back into the platform as appropriate

- As we saw earlier with Seam 2
- As we saw with AJAX and JSF 1.2

The JSF Ecosystem

Two JSF implementations:

1. Oracle's Mojarra http://java.net/projects/mojarra/
2. Apache MyFaces http://myfaces.apache.org/

JSF component suites:

1. JBoss RichFaces http://www.jboss.org/richfaces
2. IceFaces http://www.icefaces.org/
3. PrimeFaces http://www.primefaces.org/
4. Apache MyFaces Tomahawk / Trinidad / Tobago http://myfaces.apache.org/tomahawk/
5. OpenFaces http://openfaces.org/

CDI is Awesome!

CDI: Contexts and Dependency Injection for Java EE 6

Contexts: Request, Session, Conversation, etc.

Dependency Injection: @Inject, @Produces, etc.

Events: `public void myObserver (@Observes MyEvent myEvent) { ... }`

Finally! A consistent programming model for working with all parts of Java EE

JSF + CDI ⇒ Only OK.

Working with CDI and JSF 2 together is not as *Seam*less as it should be

- The scopes cannot easily interact
- Not all JSF classes are "Injectable"
- Different event mechanisms

Enter Seam Faces

Seam Faces whose goal is to:

> *Build further on the integration of JSF and CDI, and provide developers with a truly worthy web framework.*

Seam Faces aims to:

- Reduce boilerplate
- Ease integration with other technologies
- Fill in gaps in the specification

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

Colour code

JSF style “developer” code

Seam Faces style “developer” code

Seam Faces framework code

EL short-cut references, and Injection

		@Inject
`#{javax.enterprise.conversation}`	`#{conversation}`	✔
`#{facesContext}`	-	✔
`#{facesContext.externalContext}`	`#{externalContext}`	✔
`#{flash}`	-	✔
`#{facesContext.application.navigationHandler}`	-	✔
`#{facesContext.application.projectStage}`	`#{projectStage}`	✔

@Named @Produces

These EL references are made using the named CDI Producers:

```
public class ProjectStageProducer {
    @Named
    @Produces
    public ProjectStage getProjectStage(final FacesContext context) {
        return context.getApplication().getProjectStage();
    }
}
```

Conversation Boundaries

@ConversationScoped

- provided with CDI
- transient: Scope bounded by the JSF Lifecycle
- long-running: Spans multiple requests

```
@Inject Conversation conversation;

public void method() {
    conversation.begin();
    ....
}
```

Method annotations for conversation boundaries

- @Begin, @End

```
@Begin
public void method() {
    ....
}
```

<Generic> Converters

Seam Faces provides <Generic> Converters:

```
public abstract class Converter<T> implements javax.faces.convert.Converter {

    @Override
    public Object getAsObject(final FacesContext context, final UIComponent comp, final
String value) {
        this.context = context;
        return toObject(comp, value);
    }

    @Override
    public String getAsString(final FacesContext context, final UIComponent comp, final
Object value) {
        this.context = context;
        return toString(comp, (T) value);
    }

    public abstract T toObject(UIComponent comp, String value);
    public abstract String toString(UIComponent comp, T value);
}
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

DI unavailable in some JSF classes

JSF 2 has the annotation @ManagedProperty annotation for DI

```
@ManagedProperty(value="#{userManager}")
private UserManager userManager;
```

Whereas with CDI, we use the @Inject annotation

```
@Inject
private ItemServiceFacade itemService;
```

Unfortunately neither is supported out of the box throughout all JSF classes

Dependency Injection for Everyone!

JSF/CDI does not support injection in Converters & Validators

One ends up with nasty EL lookups to retrieve managed beans

```
public MyClass getMyClass() {
    ELContext context = FacesContext.getCurrentInstance().getELContext();
    MyClass myClass = (MyClass) context.getELResolver().getValue(context, null, "myClass");
    return myClass;
}
```

Seam Faces enables @Inject in JSF Converters and Validators

```
@Inject
private MyClass myClass;
```

But how?

Seam Faces implements a JSF ApplicationWrapper that intercepts all calls for Converters and Validators.

Seam Faces then provides Bean Managed instances of Converters and Validators.

```
public class SeamApplicationWrapper extends ApplicationWrapper {
    ...

    @Override
    public Converter createConverter(final String converterId) {
        log.debugf("Creating converter for converterId %s", converterId);
        Converter result = parent.createConverter(converterId);
        result = attemptExtension(result);
        return result;
    }
    ...
}
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

JSF Events

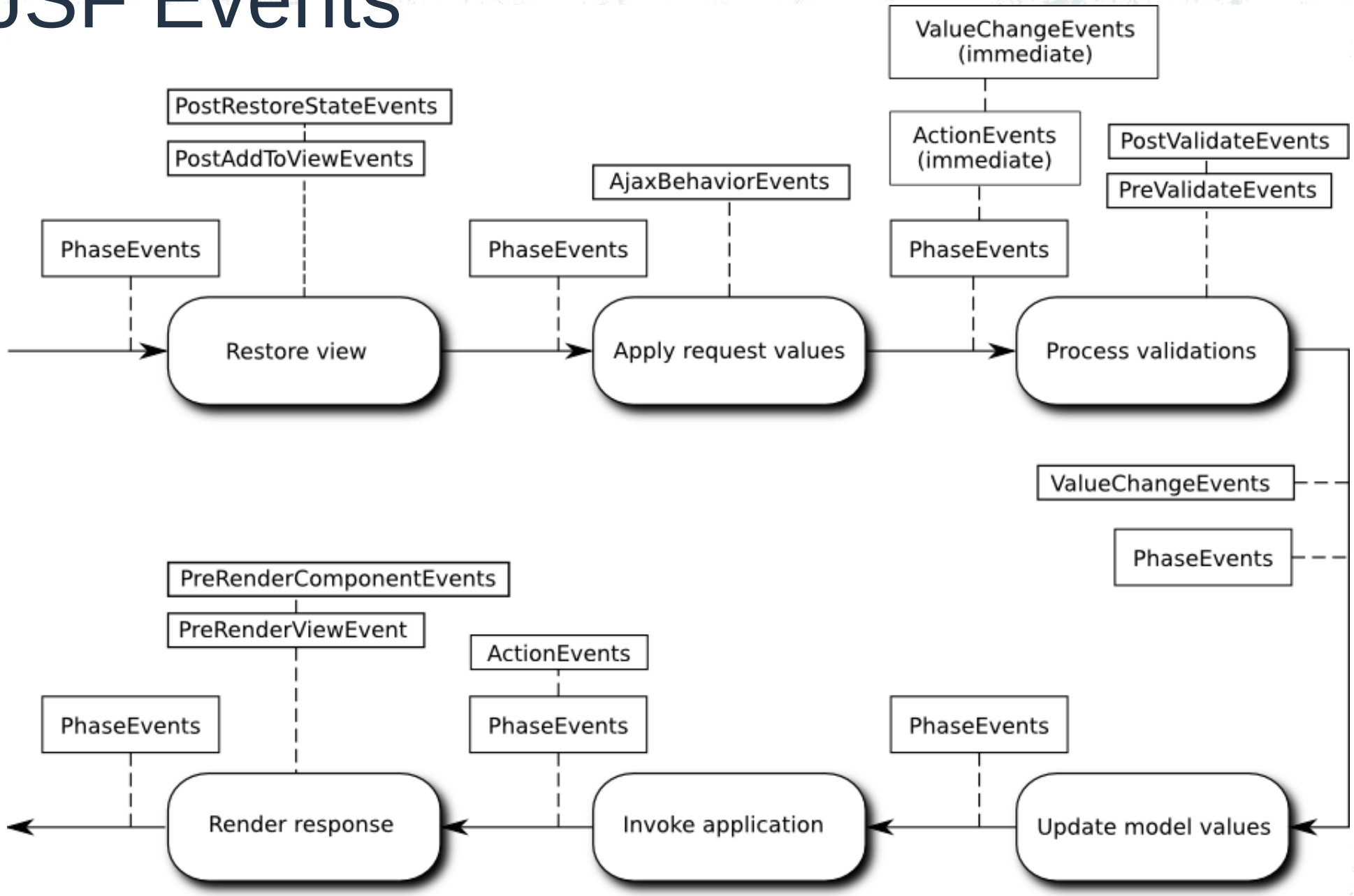

Figure: Copyright 2010 Ed Burns

JSF Phase Listeners

Implement a JSF Phase Listener - cumbersome

```
public class MyPhaseListener implements PhaseListener {

    @Override
    public void afterPhase(final PhaseEvent e) {
        // do stuff
    }

    @Override
    public void beforePhase(final PhaseEvent e) {
        // do stuff
    }

    @Override
    public PhaseId getPhaseId() {
        return PhaseId.RESTORE_VIEW;
    }
}
```

JSF System Event Listener

Implement a JSF SystemEventListener

```
public class MySystemEventListener implements SystemEventListener {

    @Override
    public boolean isListenerForSource(final Object source) {
        return true;
    }

    @Override
    public void processEvent(final SystemEvent e) throws AbortProcessingException {
        // do stuff
    }
}
```

Publish JSF Events as CDI Events

That's a lot of boiler plate!

Seam Faces writes these JSF listeners for you, and publishes the Phase Events on the CDI event bus.

Now Listening to Events is Easy!

With Seam Faces, listening to JSF events is much easier

```
public class MyClass {

    public void aMethod(@Observes @After @RestoreView PhaseEvent event) {
        // do stuff
    }

    ...

    public void bMethod(@Observes PostConstructApplicationEvent event) {
        // do stuff
    }
}
```

Additional events

Seam Faces adds to the already rich list of JSF events available:

`@Observes PreNavigate event`

- useful for intercepting and altering JSF navigation

`@Observes PreLogin event`
`@Observes PostLogin event`

- Provide a session map where values can be stored/retrieved during the login process

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

View Configuration

Seam Faces offers a type-safe mechanism to configure the behaviour of your JSF views.

So far these configurable behaviors include:

1. Restricting view access by integrating with Seam Security
2. Configuring URL rewriting by integrating with PrettyFaces (or any other pluggable URL rewriting framework
3. Configuring Transactions via Seam Persistence
4. And a personal favorite: setting “?faces-redirect=true” when navigating to a view

@ViewConfig – configuration enum

```
@ViewConfig
public interface MyAppViewConfig {

    static enum Pages {

        @ViewPattern("/admin.xhtml")
        @Admin
        ADMIN,

        @ViewPattern("/item.xhtml")
        @UrlMapping(pattern="/item/#{id}/")
        @Owner
        ITEM,

        @ViewPattern("/*")
        @FacesRedirect
        @AccessDeniedView("/denied.xhtml")
        @LoginView("/login.xhtml")
        ALL;

    }
}
```

@ViewConfig annotations

```
@ViewPattern("/*")
@FacesRedirect
@AccessDeniedView("/denied.xhtml")
@LoginView("/login.xhtml")
ALL;
```

`@FacesRedirect`

sets faces-redirect=true for all associated JSF navigations

`@LoginView`

the view to navigate to when login is required

`@AccessDeniedView`

the view to navigate to when access is denied

@ViewConfig - Securing Views

Use Annotations to link view patterns to Securing methods

```
@ViewPattern("/item.xhtml")
@UrlMapping(pattern="/item/#{id}/")
@Owner
ITEM,

...

@SecurityBindingType
@Retention(RetentionPolicy.RUNTIME)
public @interface Owner {
}

...

public @Secures @Owner boolean ownerChecker(Identity identity, @Current Item item) {
    if (item == null || identity.getUser() == null) {
        return false;
    } else {
        return item.getOwner().equals(identity.getUser().getId());
    }
}
```

@ViewConfig - URL Rewriting

Use Annotations to rewrite URLS

```
@ViewPattern("/item.xhtml")
@UrlMapping(pattern="/item/#{id}/")
@Owner
ITEM,
```

“#{id}” tells the rewriting-engine to treat the last portion of the URL as a the value of the query-parameter named “id”

urls like /item.jsf?item=1 get mapped into /item/1/

Courtesy of PrettyFaces (http://ocpsoft.com/prettyfaces/)

The View Config Store

An API to programatically add and remove view configuration from the data store.

Used by Seam Faces extensions to read the annotation from the view config

The plan is to provide additional ways to feed data into this store:

- <f:metadata> child tags
- persistent database store

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

Exception Handling

Java Applications throw exceptions. Fact of life.

Applications need to properly handle exceptions, provide appropriate logging, and direct the user accordingly.

Handling Exceptions in JSF is ugly.

- Handled properly with using a JSF ExceptionHandler
- Lots of boiler plate code

JSF ExceptionHandlerFactory

```
public class MyHandlerFactory extends ExceptionHandlerFactory {

    public MyHandlerFactory(ExceptionHandlerFactory parent) {
        super();
        this.parent = parent;
    }

     @Override
    public ExceptionHandler getExceptionHandler() {
        ...
    }
}
```

```
<factory>
   <exception-handler-factory>
      my.package.MyHandlerFactory
   </exception-handler-factory>
</factory>
```

JSF ExceptionHandler

```
public class MyHandler extends ExceptionHandlerWrapper {

    @Override
    public ExceptionHandler getWrapped() {
        return this.wrapped;
    }

    @Override
    public void handle() throws FacesException {
        ...
    }
}
```

Integration with Seam Catch

Seam Faces publishes all JSF Exceptions using the CDI event mechanism
One writes Catch Exception Handlers to handle the exceptions

```
@HandlesExceptions
public class MyHandlers {
   void printExceptions(@Handles CaughtException<Throwable> evt) {
       System.out.println("Something bad happened: " + evt.getException().getMessage());
      evt.markHandled();
    }
}
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

Non-visual Components

<s:viewAction>

<s:validateForm>

<sc:inputContainer>

<s:viewAction> preRenderView++ !

```
<f:metadata>
    <f:viewParam name="id" value="#{itemManager.itemId}"/>
    <f:event name="preRenderView" listener="#{itemManager.loadEntry}" />
</f:metadata>
```

Doesn't perform navigation

Can't conditionally choose the execution phase

Can't disable for a POST

```
@Named
@RequestScoped
public class ItemManager {
    private Item itemId;

    public Item getItemId() {
        return itemId;
    }

    public void setItemId(Integer id) {
        ItemId = id;
    }

    private void loadItem() {
       // do stuff
   }
}
```

<s:viewAction>

A view action is a UICommand for an initial (non-faces) request.

```
<f:metadata>
   <f:viewParam name="id" value="#{itemManager.itemId}"/>
   <s:viewAction action="#{itemManager.loadItem}"/>
</f:metadata>
```

Can perform navigation - a full fledged UICommand

Can conditionally choose the execution phase

Can disable for a POST

<s:validateForm>

Perform cross-field form validation is simple
Place the <s:validateForm> component in the form you wish to validate, then attach your custom Validator.

```
 <h:form id="locationForm">
     <h:inputText id="cityId" value="#{bean.city}" />
     <h:inputText id="stateId" value="#{bean.state}" />
     <h:inputText id="zip" value="#{bean.zip}" />
     <h:commandButton id="submit" value="Submit" action="#{bean.submitPost}" />

     <s:validateForm fields="city=cityId state=stateId" validatorId="locationValidator" />
 </h:form>


@FacesValidator("locationValidator")
public class MyValidator implements Validator {

    @Inject
    private InputElement<String> city;
    @Inject
    private InputElement<String> state;
    ...
}
```

<sc:inputContainer>

Wraps any input component (EditableValueHolder)

```
xmlns:sc="http://java.sun.com/jsf/composite/components/seamfaces"

<sc:inputContainer label="name" id="name">
    <h:inputText id="input" value="#{person.name}"/>
</sc:inputContainer>
```

Reduces Facelet boiler plate, by creating:

- The JSF label, with required flag
- The message associated with the input
- matches up the "id" and the "for" attributes as required

<sc:inputContainer>

Define your own InputContainer to control the layout

```
<cc:interface componentType="org.jboss.seam.faces.InputContainer">
   <cc:attribute name="label" required="true"/>
   <cc:attribute name="required" required="false"/>
   <cc:attribute name="ajax" required="false" default="false"/>
   <cc:attribute name="inputs" required="false" default="1"/>
</cc:interface>
<cc:implementation>

   <div class="entry" id="#{cc.clientId}">
      <h:outputLabel id="label" for="" value="#{cc.attrs.label}:"
                     styleClass="#{cc.attrs.invalid ? 'label errors' : 'label'}">
         <h:panelGroup styleClass="required" rendered="#{cc.attrs.required}">*</h:panelGroup>
      </h:outputLabel>
      <span class="#{cc.attrs.invalid ? 'input errors' : 'input'}">
         <cc:insertChildren/>
      </span>
      <h:panelGroup rendered="#{cc.attrs.invalid}">
         <c:forEach var="i" begin="1" end="#{cc.attrs.inputs}">
            <h:message id="message#{i}" for="" styleClass="error errors"/>
         </c:forEach>
      </h:panelGroup>
   </div>

</cc:implementation>
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

CDI Scopes

Override the JSF Scopes with CDI enabled scopes

- javax.faces.bean.RequestScoped
  javax.enterprise.context.RequestScoped
- javax.faces.bean.SessionScoped
  javax.enterprise.context.SessionScoped
- javax.faces.bean.ApplicationScoped
  javax.enterprise.context.ApplicationScoped
- javax.faces.bean.ViewScoped
  // No CDI equivalent

@RenderScoped

- *Beans placed in the @RenderScoped context are scoped to, and live through but not after, the next "Render Response" phase*

Upcoming features

Killer Scope

Type safe navigation rules

@ViewConfig controls in <f:metadata>

Global Protection against XSRF attacks

JSF 2.2

The next iteration of the JSF spec is being hashed out.

CDI Integration is on the table:
http://java.net/jira/browse/JAVASERVERFACES_SPEC_PUBLIC-976

Conclusion

Seam Faces improves upon the JSF 2 / CDI integration, providing JSF developers with a full featured framework for developing Web Applications

- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- CDI Scopes

Get Involved

Get involved with Seam Faces, and help make the JSF platform into what you need it to be.

http://seamframework.org/Seam3/FacesModule
http://seamframework.org/Community

Twitter: #SEAM, #SEAMFACES
http://twitter.com/brianleathem

Github: https://github.com/seam/faces