



JBoss Seam

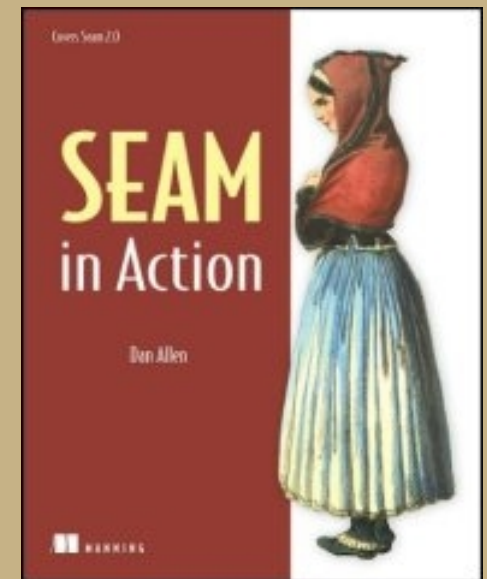
Integration with intent to use

Dan Allen
Software Consultant
Author, Seam in Action



Who am I?

- Author of Seam in Action
- Author of the *Seamless JSF* series
- Committer on the JBoss Seam Project
- Software consultant
- Linux and open source advocate
- <http://www.mojavelinux.com>



Who am I *really*?

A software developer looking for solutions to deliver projects **on time** and **on budget**

...and who digs



What is JBoss Seam?

→ I've overheard some say...

- ⊙ "Seam, that sounds too technical for me."
- ⊙ "Isn't SEAM an acronym for something?"
- ⊙ "Can Seam finish my curtains?"



What is JBoss Seam?

Let's not start with this question

Let's **start** with the *problem*
...then **look** for a *solution*



Dan's Top 5 List: Development challenges

- #5 You have a lot of requirements, but not a lot of time
- #4 You burn too much time setting up the project
- #3 You have to integrate many disparate technologies
- #2 Managing state in a web application is a *pain*
- #1 You never get to the fun stuff (Ajax, PDFs, email)



Seam as a solution

Let's see how Seam solves our problems



Seam-gen: Seam's project generator

- Assembles Ant-based project in minutes
 - Supports EAR and WAR archive formats
- Ready to be imported into an IDE
 - Officially supports Eclipse and NetBeans
 - Any IDE that can drive Ant tasks
- Can handle deployment to JBoss AS
 - Incremental hot redeployment for "instant change"
- Three build profiles (dev, test, prod)



Why use seam-gen?

- Let's you get to work quickly and be productive
- Let's you use Seam in its *natural* environment
- Gets you doing the fun stuff sooner

You can always go your own way later



Seam-gen: Seam's code generator

- Seam's answer to Ruby on Rails (and Grails)
- Can build an entire CRUD application
 - From a database
 - From existing Java classes
- Can generate code templates
 - View templates (Facelets)
 - JPA entity classes
 - Action components
 - Integration test cases



Seam-gen: setup

- Requirements to setup new project
 - ⊙ Seam distribution (includes seam-gen)
 - ⊙ JBoss AS \geq 4.2.2 installed
 - ⊙ Database server (unless using embedded)
 - ⊙ Database JDBC driver JAR
 - ⊙ Existing tables unless generating schema
 - ⊙ Project name and directory



Multilayer living

- Persistence layer
- Business layer
- Presentation layer

Every web application's got 'em
It's your job to *integrate* them



The Seam component model

- Unifies components across layers/technologies
 - ⊙ Java Persistence API (JPA) or Hibernate
 - ⊙ Enterprise JavaBeans (EJB 3) or plain JavaBeans
 - ⊙ JavaServer Faces (JSF)
- Unifies contexts across entire application
 - ⊙ Java Servlet API (request, session, application)
 - ⊙ JSF component tree (page)
 - ⊙ Seam (conversation)
 - ⊙ jBPM (business process)



Contextual components

→ Component

- Definition of how to create an object
- Managed class instantiated by Seam
- Instances represent state of system; typically scoped to the use-case

→ Context

- Buckets where the component instances are stored
- Accessible from anywhere
- One way to access contexts

Contextual means that **state** matters!



Seam component > JSF managed bean

- Shares same idea of a container-created object
- JSF can also store object in stateful scopes
 - Scope options are limited
- Seam component is more capable
 - Additional scope options
 - Strung with interceptors
 - manages life cycle of instance
 - decorates with services (transactions, security, etc)



JSF managed bean

```
<managed-bean>                                     faces-config.xml
  <managed-bean-name>beerAction</managed-bean-name>
  <managed-bean-class>
    org.connessieur.action.BeerAction
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```



JavaBean-Seam component

```
@Name("beerAction")  
public class BeerAction {  
    ...  
}
```

- Stored in event context by default
 - ⊙ Event context == request scope



Conversational component

```
@Name("beerAction")  
@Scope(ScopeType.CONVERSATION)  
public class BeerAction {  
    ...  
}
```

- Seam introduces the conversation context
 - Supports a single use-case, which may span multiple pages and actions



EJB 3-Seam component

```
@Stateful  
@Name("beerAction")  
public class BeerActionBean implements BeerAction {  
    ...  
}
```

- Instantiated by EJB 3 container then intercepted by Seam
- Stored in conversation context by default



Annotations instead of XML

No more XM-hell!



More concise XML

But if you must...

```
<component name="beerAction"  
  class="org.connessieur.action.BeerAction"/>
```

- Declared in the Seam component descriptor
 - ⊙ META-INF/components.xml
 - ⊙ BeerAction.component.xml



Looking up a component

- Every component has a name -> beerAction
- When name is requested, an instance is created
 - Instance is stored in a *context* variable
- Several ways to perform lookup
 - EL value expression -> `# {beerAction}`
 - Annotation -> `@In("beerAction")`
 - Seam API -> `Component.getInstance("beerAction")`



Using a component

- Capture form data
 - Bind properties to input fields
- Respond to action in user interface
 - Click a button
 - Submit a form
- Model a sequence of interactions
 - Demarcate conversation boundaries
 - Hold state for a conversation



Seam-gen: new-form

- Use new-form command to create stub JSF view
 - ⊙ Component name: `beerAction`
 - ⊙ Bean class name: `BeerAction`
 - ⊙ Action method name: `save`
 - ⊙ Page name: `editBeer`
- Artifacts generated
 - ⊙ `BeerAction.java` (under `src/action`)
 - ⊙ `editBeer.xhtml` (under `view`)
 - ⊙ `BeerActionTest.java` (under `src/test`)



Components as action listeners

- Component responds to user generated event
 - button, link, input value change
 - action triggers method execution on server
- Action listener registered using method binding expression



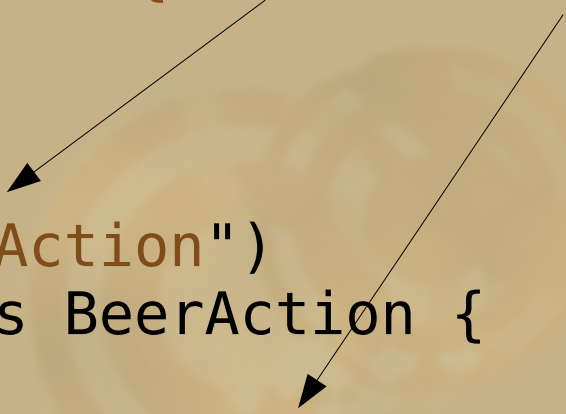
Binding the action listener

```
<h:form id="beer">
```

editBeer.xhtml

```
    ...  
    <div class="actionButtons">  
        <h:commandButton id="save" value="Save"  
            action="#{beerAction.save}"/>  
    </div>  
</h:form>
```

```
@Name("beerAction")  
public class BeerAction {  
    public String save() { ... }  
}
```



Entity class as a component

- Central piece in object-relational mapping (ORM)
 - Transports data to/from database
- Also plays central role in Seam application
 - Shared across the layers
 - Can serve as a JSF "backing" form bean
- Life cycle of entity controlled by ORM
 - Not managed by Seam like other components
 - Seam seeds the new instance (prototype)



Defining an entity component

```
@Entity
@Name("beer")
public class Beer implements Serializable {
    protected Long id;
    protected String name;
    ...
    @Id @GeneratedValue
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String n) { this.name = n; }
    ...
}
```



Capturing form input in JSF

- Abstraction over HTTP protocol and POST data
- Input element bound directly to model property
 - ⊙ Uses value binding expression -> `#{beer.name}`
- On submit, JSF will...
 - ⊙ read value from POST data -> `name`
 - ⊙ convert and validate value
 - ⊙ create new entity class instance -> `Beer`
 - ⊙ assign value to property on model -> `setName()`



Binding entity to form inputs

```
<h:form id="beer">
    <rich:panel>
        <f:facet name="header">Add Beer</f:facet>
        <s:decorate id="nameField"
            template="layout/edit.xhtml">
            <ui:define name="label">Name</ui:define>
            <h:inputText id="name" required="true"
                value="#{beer.name}"/>
        </s:decorate>
        ...
    </rich:panel>
</h:form>
```

editBeer.xhtml

Add Beer

Name *

Beer style *

Brewer *

Official Site


Description *

* required fields



Seam UI decorator

- Builds JSF markup dynamically
 - Composite view pattern
- Adds two implicit context variables
 - **required** – as defined on input component
 - **invalid** – indicates if input has validation errors
- layout/edit.xhtml ties label and validation error message to input component

Name *	<input type="text" value="Hop Wallop"/>	
Beer style *	<input type="text" value="a"/>	 length must be between 3 and 25



Enforcing validations

- Before submitting form, inputs must be validated
- Validations traditionally defined using UI component tags
 - `<f:validateLength minimum="3" maximum="50"/>`
- Better to define validations in domain model
 - Not repeated on every page property is used
 - Centralized for other layers to access



Hibernate validator

- Criteria defined on property using annotations
 - ⊙ @Length(min = 3, max = 50)
 - ⊙ @Email
 - ⊙ @CreditCardNumber
- Seam adapts Hibernate validator to JSF validation
 - ⊙ <s:validate/> nested within input component
 - ⊙ <s:validateAll> wrapped around all inputs
 - used in layout/edit.xhtml
- Validations take effect in UI immediately
 - ⊙ Only worth is that required="true" still necessary

Model validations

```
@Entity
@Name("beer")
public class Beer implements Serializable {
    ...
    protected String name;

    @Length(min = 3, max = 50)
    public String getName() { return this.name; }
    ...
}
```



Ready to save?

Test first!



Seam's testing infrastructure

- Seam provides a test infrastructure for end-to-end testing
 - Based on TestNG
 - Boots an embedded Java EE environment
 - Most seam-gen commands create a test stub
- Instantiate JSF life cycle inside of test case
 - NonFacesRequest – emulates initial "GET" request
 - FacesRequest – emulates JSF "postback"
- Execute tests using `ant test`



Defining the expected behavior

```
public class BeerActionTest extends SeamTest {
    @Test public void testAddBeer() throws Exception {
        new FacesRequest("/editBeer.xhtml") {
            protected void updateModelValues()
                throws Exception {
                setValue("#{beer.name}", "HopDevil");
            }
            protected void invokeApplication()
                throws Exception {
                invokeMethod("#{beerAction.save}");
            }
        }.run();
    }
}
```



Implementing the behavior

```
@Name("beerAction")
public class BeerAction {

    protected Beer beer;

    protected EntityManager entityManager;

    public String save() {
        entityManager.persist(beer);
        return "success";
    }
}
```



Bijection

- Similar to dependency injection
 - ⦿ container satisfies the needs of components
- Bijection occurs on *every* method invocation
- Prefix "bi" implies that it occurs twice
 - ⦿ *injection* is applied before the method is invoked
 - ⦿ *outjection* is applied after the method is invoked

Bijection = Injection + Outjection



Dynamic @In-jection

- @In annotation applied to property of class
- Simple case
 - Seam looks for a context variable matching the name of the property
 - assigns value of context variable to property
- Additional criteria
 - customize context variable name and scope
 - set required=false to ignore missing value
 - set create=true to create value if not found



@Out-jection: variable export

- @Out annotation applied to property of class
- Simple case
 - Value of property is assigned to a context variable whose name matches the name of the property
 - Scope of context variable determined as follows:
 - use scope of component with same name, if exists
 - otherwise, use scope of property's component
- Additional criteria
 - customize context variable name and scope
 - set `required=false` to ignore null value



What bijection buys you

- Simple to wire together objects and tap into managed services
- It's dynamic
 - Can inject a narrower-scoped component into a wider-scoped component
 - properties are updated to reflect latest state

No manual assignment to make variables available to the view



Using bijection to implement the behavior

```
@Name("beerAction")
public class BeerAction {
    @In ←
    protected Beer beer; ← inject context variables

    @In ←
    protected EntityManager entityManager;

    public String save() {
        entityManager.persist(beer);
        return "success";
    }
}
```



Using bijection to implement the behavior

```
@Name("beerAction")
public class BeerAction {
    @In
    protected Beer beer;

    @In
    protected EntityManager;

    public String save() {
        entityManager.persist(beer);
        FacesMessages.instance().add("#{beer.name}");
        return "/home.xhtml";
    }
}
```

issues "redirect after post"
but JSF message is not lost



Where is the EntityManager defined?

- Seam automatically creates a new EntityManager on demand from JPA persistence unit
- Defined in components.xml
 - XML-based equivalent to @Name

```
<persistence:managed-persistence-context  
  name="entityManager" auto-create="true"  
  entity-manager-factory="#{entityManagerFactory}"/>
```

```
<persistence:entity-manager-factory  
  name="entityManagerFactory"  
  persistence-unit-name="connoisseur"/>
```



What about transactions?

- Seam wraps every JSF request in two transactions
 - ...from start of request until after actions are invoked (*Invoke Application* phase)
 - ...around the rendering of the response (*Render Response* phase)
- Works with both JTA and entity transactions
- @Transactional marks a method as transactional
 - Don't need it as long as using JSF life cycle
 - Is required if you disable Seam global transactions



Context variables on demand

- In component-based frameworks, data is "pulled" rather than "pushed"
- Instance of component is created when name is requested
- What about arbitrary data?
- Seam factory component
 - Provides a way to initialize *any* context variable when requested



Pulling down a list

```
<rich:panel> beerList.xhtml
  <f:facet name="header">Beers of the world</f:facet>
  <rich:dataTable var="_beer" value="#{beers}"
    rendered="#{not empty beers}">
    <h:column>
      <f:facet name="header">Name</f:facet>
      #{_beer.name}
    </h:column>
    ...
  </rich:dataTable>
</rich:panel>
```

Beers of the World		
Name	Style	Brewer
HopDevil Ale	American IPA	Victory Brewing Company
Troegs Pale Ale	American Pale Ale	Troegs Brewing Company
Nugget Nectar Ale	American Amber	Troegs Brewing Company



A context variable factory

```
@Name("beerList")
public class BeerList {
    @In
    protected EntityManager entityManager;

    @Factory("beers")
    public List<Beer> getBeers() {
        return entityManager
            .createQuery("select b from Beer b")
            .getResultList();
    }
}
```



Factory chain reaction

```
@Name("beerList")
public class BeerList {
    @In
    protected EntityManager entityManager;

    @Out ← exported after factory method is called
    protected List<Beer> beers;

    @Factory("beers")
    public void loadBeers() {
        beers = entityManager
            .createQuery("select b from Beer b")
            .getResultList();
    }
}
```



Data model selection

- JSF UIData components support notion of row selection
 - ⊙ `<h:dataTable>`
- Captured when action is triggered within row
- How does it work?
 - ⊙ JSF `DataModel` wraps collection
 - ⊙ JSF positions `DataModel#getRowData()` before calling action listener
 - ⊙ Action listener must have reference to `DataModel` to access selected row data :(



Transparent data model selection with Seam

```
@Name("beerList")
public class BeerList {
    ...
    @DataModel(scope = ScopeType.PAGE)
    protected List<Beer> beers;
    @DataModelSelection
    protected Beer selectedBeer;

    @Factory("beers")
    public void loadBeers() { beers = ...; }

    public String processSelection() { ... }
}
```

no direct use of DataModel



"Clickable" lists

```
<rich:dataTable var="_beer" value="#{_beers}" beerList.xhtml
  rendered="#{_beers.rowCount gt 0}">
  ...
  <h:column>
    <f:facet name="header">Action</f:facet>
    ...
    <h:commandLink action="#{beerList.delete}"
      value="Delete"/>
  </h:column>
</rich:dataTable>
```

Beers of the World			
Name	Style	Brewer	Action
HopDevil Ale	American IPA	Victory Brewing Company	Edit Delete
Troegs Pale Ale	American Pale Ale	Troegs Brewing Company	Edit Delete
Nugget Nectar Ale	American Amber	Troegs Brewing Company	Edit Delete



Processing the selected row

```
@Name("beerList")
public class BeerList {
    ...
    @DataModelSelection
    protected Beer selectedBeer;
    ...
    public String delete() {
        selectedBeer = entityManager
            .find(Beer.class, selectedBeer.getId());
        entityManager.remove(selectedBeer);
        return "/beerList.xhtml";
    }
}
```



Handing off a selection

```
<rich:dataTable var="_beer" value="#{_beers}" beerList.xhtml
  rendered="#{_beers.rowCount gt 0}">
  ...
  <h:column>
    <f:facet name="header">Action</f:facet>
    <s:link view="/editBeer.xhtml"
      action="#{beerAction.edit}" value="Edit">
      <f:param name="id" value="#{_beer.id}"/>
    </s:link>
    ...
  </h:column>
</rich:dataTable>
```

Beers of the World			
Name	Style	Brewer	Action
HopDevil Ale	American IPA	Victory Brewing Company	Edit Delete
Troegs Pale Ale	American Pale Ale	Troegs Brewing Company	Edit Delete
Nugget Nectar Ale	American Amber	Troegs Brewing Company	Edit Delete



Selecting a row for editing

```
@Name("beerAction")
public class BeerAction {
    ...
    @In(create = true) protected Beer beer;
    @RequestParam protected Long id;
    ...
    public String edit() {
        beer = entityManager.find(Beer.class, id);
        return "/editBeer.xhtml";
    }
}
```



Checking out a record

- Need to track record's state
 - Create or edit mode
 - Unique id of record being modified, if edit mode
 - Lock status
- Traditionally done using hidden form fields
 - Entity must be reloaded from database
 - Form data must be copied onto entity instance



Persistence context

- Reference to retrieved entities
 - Maintained by persistence manager
 - In memory cache
 - Instances are "managed"
 - Guarantees identity of instances
 - Performs automatic dirty checking
- *All that stops working when closed!*
 - Entity instances become "detached"



Extended persistence context

- Persistence manager stored in conversation
- Lives across requests
- JSF applies form values to *managed* entity
- Dirty checking ensures update
- Automatic optimistic locking



Starting the conversation

```
@Name("beerAction")
@Scope(ScopeType.CONVERSATION)
public class BeerAction {
    ...
    @In(create = true) @Out protected Beer beer;
    @RequestParam protected Long id;
    @Out protected boolean managed;

    @Begin
    public String edit() {
        beer = entityManager.find(Beer.class, id);
        managed = true;
        return "/editBeer.xhtml";
    }
}
```



Save or update?

```
<h:form id="beer">
```

editBeer.xhtml

```
...
```

```
<div class="actionButtons">
```

```
<h:commandButton action="#{beerAction.save}"  
  rendered="#{not managed}" />
```

```
<h:commandButton action="#{beerAction.update}"  
  rendered="#{managed}" />
```

```
</div>
```

```
</h:form>
```



Wrapping things up

```
@Name("beerAction")  
@Scope(ScopeType.CONVERSATION)  
public class BeerAction {
```

```
    ...
```

no explicit update instruction

```
    @End
```

```
    public String update() {  
        managed = false;  
        FacesMessages().instance()  
            .add("#{beer.name} has been updated.");  
        return "/editBeer.xhtml";  
    }  
}
```



Multi-record editing

```
@Name("beerList")
@Scope(ScopeType.CONVERSATION)
public class BeerList {
    ...
    @Out protected boolean editModeEnabled;

    @Begin
    public void editMode() {
        loadBeers(); ← load records into
        editModeEnabled = true; persistence context
    }
    ...
}
```



Multi-record editing

```
<rich:dataTable var="_beer" value="#{beers}"> beerList.xhtml
  <h:column>
    <f:facet name="header">Name</f:facet>
    <h:outputText value="#{_beer.name}"
      rendered="#{not editModeEnabled}" />
    <h:inputText value="#{_beer.name}" required="true"
      rendered="#{editModeEnabled}" />
  </h:column>
  ...
</rich:dataTable>
```

Beers of the World

Multi-edit mode **enabled**.

Name	Style	Brewer	Official Site
<input type="text" value="HopDevil Ale"/>	<input type="text" value="American IPA"/>	<input type="text" value="Victory Brewing Company"/> ▼	<input type="text"/>
<input type="text" value="Troegs Pale Ale"/>	<input type="text" value="American Pale Ale"/>	<input type="text" value="Troegs Brewing Company"/> ▼	<input type="text" value="http://www.troegs.com/"/>
<input type="text" value="Nugget Nectar Ale"/>	<input type="text" value="American Amber"/>	<input type="text" value="Troegs Brewing Company"/> ▼	<input type="text"/>



Multi-record editing

```
@Name("beerList")
@Scope(ScopeType.CONVERSATION)
public class BeerList {
    ...
    @Out protected boolean editModeEnabled;

    @End
    public void saveChanges() {
        editModeEnabled = false;
        FacesMessages.instance().add("Changes saved");
        return "/beerList.xhtml";
    }
    ...
}
```



And there's more...

Please log in first

Login

Please login here

Username

Password

Remember me

Authentication

Login

HopDevil Ale
American IPA
by Victory Brewing Company

Description

It's **menacingly** delicious, with the powerful, aromatic punch of whole flower American hops backed up by rich, German malts. HopDevil Ale offers a roller coaster ride of flavor, coasting to a smooth finish that satisfies fully.

PDF creation

Troegs Pale Ale	American Pale Ale	Troegs Brewing Company	Edit
-----------------	-------------------	------------------------	------

Nectar **Search** *Hibernate Search*

Beers of the World

Name	Style	Brewer	Action
Nugget Nectar Ale	American Amber	Troegs Brewing Company	Edit Delete

Add Beer

Name *

Beer style *

Brewer *

- American Amber
- American IPA
- American Pale Ale**

Official Site

Description *

* required fields

Form autocomplete

Tooltips

Description - Troegs Pale Ale

A Troegs Brewery classic, our Pale Ale is copper colored with generous amounts of Cascade hops to create a floral, aromatic pale ale that smells as delicious as it tastes.



**Questions?
I know you've got 'em**

Now it's your turn to challenge me!



Resources

- Seam 2.0 books
 - Seam in Action, by Dan Allen
- Seam community site
 - <http://seamframework.org>
- Seam Issue Tracker
 - <http://jira.jboss.org/jira/browse/JBSEAM>
- Seam links (and lots of them)
 - <http://del.icio.us/seaminaction>



Thanks for coming!

Dan Allen

dan.allen@mojavelinux.com

<http://www.mojavelinux.com>

