DZone Refcardz

Brought to you by... JBoss by Red Hat

CONTENTS INCLUDE:

Seam UI

By Jacob Orshalick

ABOUT SEAM UI

Seam is a next generation web framework that integrates standard Java EE technologies with a wide variety of non-standard technologies into a consistent, unified, programming model. Seam drove the development of Web Beans (JSR-299) and continues to develop innovations that are changing the face of web development as well as Java EE technologies. If you haven't taken a look at Seam, I suggest you do.

As you develop Seam applications, you'll find this reference a handy guide for understanding how Seam simplifies JSF development. This reference does not cover all of what Seam provides, but covers the most commonly used UI annotations and XML elements as of Seam 2.1.

SIMPLIFYING JSF

While Seam is no longer limited to JSF as it's view layer, one of the framework's initial goals was simplifying JSF development. Seam plugs into the JSF lifecycle to provide a number of enhancements including:

- Direct Facelets support (https://facelets.dev.java.net/)
- A unified component model
- Additional contexts including conversations
- Bean validations
- A component event model
- Exception handling
- RESTful URLs
- Extended EL support
- And much more...

PAGE NAVIGATION

Page navigation with JSF can be verbose and only provides a limited feature set. Seam provides a concise navigation language that incorporates conversation management, security, exception handling, request parameters, event notification, and more. Here we will discuss the pages namespace which defines the navigation flow of a Seam application as well as the navigation namespace for configuring the navigation components.

The Pages Namespace

Schema URI

http://jboss.com/products/seam/pages

Schema XSD

http://jboss.com/products/seam/pages-2.1.xsd

The pages namespace is used to define the `WEB-INF/pages.xml` definition. The `pages.xml` file unifies navigation logic

Page Navigation, continued

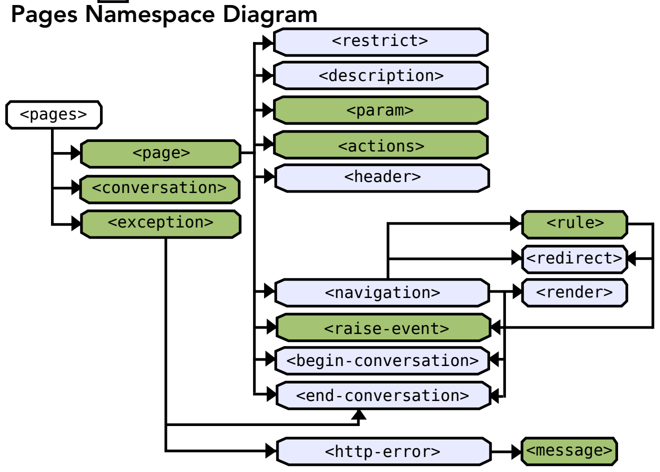
with Seam "orchestration" logic. This logic includes page parameters, conversation management, and allows navigation to be based on evaluation of arbitrary EL expressions instead of relying on return values. In addition, event notification is possible, generic exception handling, security restrictions, etc.

Seam XML Diagram Key

The Seam XML diagrams use the following notations to indicate required elements, cardinality, and containment:

□	Required XML Element	0..* 🟩	Zero or more
0..1 □	Zero or one	→	Containment

Pages Namespace Diagram



Pages Namespace Elements

Element	Description
<pages>	Root element defining general configuation applied to all pages.
<page>	Defines the navigation as well as "orchestration" logic associated with a view-id.

Page Navigation, continued

`<conversation>`	Configures a named natural conversation.
`<exception>`	Defines handling for a specific type of exception.
`<http-error>`	Specifies an HTTP error code to be returned to the user.
`<message>`	Specifies a message that should be displayed to the user.
`<restrict>`	Configures a security restriction for a page defined through EL.
`<description>`	Provides a description for a page that will be displayed in the ConversationSwitcher. Note that without this description the conversation will not show up in this component.
`<param>`	Defines a page parameter to be set from a GET request query parameter into a component.
`<action>`	Defines an action to be executed as a method-binding expression.
`<header>`	Specifies HTTP headers to be added to a page.
`<navigation>`	Defines the navigation rules associated with a page.
`<rule>`	Specifies a specific action outcome or boolean value-binding expression under which navigation should occur.
`<redirect>`	Redirects the user to the specified view-id.
`<render>`	Specifies a view-id to be rendered.
`<raise-event>`	Configures an event to be raised on page display or navigation.
`<begin-conversation>`	Begins a long-running conversation either on access of a page or on navigation.
`<end-conversation>`	Ends a long runing conversation either on access of a page, on navigation, or on occurrence of an exception.

Pages Namespace Examples

Defining Navigation Rules

The following example defines a few pages with some simple navigation rules based on the execution of EL expressions:

Pages Namespace Examples

```
<pages xmlns=”http://jboss.com/progucts/seam/pages”
    xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
    xsi:schemaLocation=
      “http://jboss.com/products/seam/pages
       http://jboss.com/products/seam/pages-2.1.xsd”
    no-conversation-view-id=”/main.xhtml”
    login-view-id=”/home.xhtml”>
  ... ...
  <page view-id=”/main.xhtml” login-required=”true”>
    <navigation from-action=
        ”#{hotelBooking.selectHotel(hotel.id)}”>
      <begin-conversation />
      <redirect view-id=”/hotel.xhtml”/>
    </navigation>
    ... ...
  </page>
  <page view-id=”/rewards.xhtml”>
    <restrict>#{s:hasRole(‘REWARDS’)}</restrict>
  </page>
  ... ...
  <exception class=
      “org.jboss.seam.security.AuthorizationException”>
    <end-conversation/>
    <redirect view-id=”/generalError.xhtml”>
      <message>You are not authorized</message>
    </redirect>
  </exception>
  ... ...
</pages>
```

When a hotel is selected on the `/main.xhtml` view and the `selectHotel` method is invoked, we begin a long-running conversation and the user is redirected to the `/hotelxhtml` view. A user is required to be logged in to access `/main.xhtml.` If the user is not logged in, he or she will be re-directed to the `login-view-id`, or `/home.xhtml` in this case.

The `/rewards.xhtml` page is further restricted to users with the `REWARDS` role. The `s:hasRole` EL function throws an `AuthorizationException` if the user is not authorized thereby ending the conversation and sending the user to the error page with an appropriate message.

Using Page Parameters and Natural Conversations

The following example defines a natural conversation named `Booking`. The `parameter-name` and `parameter-value` define the parameter that will be using to uniquely identify a conversation instance. You must ensure that the EL expression evaluates to a value when the conversation is initialized.

Using page Parameters and Natural Conversations

```
<pages xmlns=”http://jboss.com/products/seam/pages”
    xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
    xsi:schemaLocation=
      “http://jboss.com/products/seam/pages
       http://jboss.com/products/seam/pages-2.1xsd”
    no-conversation-view-id=”/main.xhtml”
    login-view-id=”/home.xhtml”>
    ... ...
  <conversation name=”Booking” parameter-name=”hotelId”
      parameter-value=”#{hotel.hotelId}” />
  <page view-id=”/hotel.xhtml” conversation=”Booking”
      login-required=”true”>
    <param name=”hotelId” value=”#{hotelBooking.hotelId}” />
    <begin-conversation join=”true” />
    ... ...
  </page>
  <page view-id=”/book.xhtml” conversation=”Booking”
      conversation-required=”true” login-required=”true”>
    ... ...
  </page>
  <page view-id=”/confirm.xhtml” conversation=”Booking”
      conversation-required=”true” login-required=”true”>
    ... ...
  </page>
</pages>
```

Each of the pages defined participate in the natural conversation by specifying `Booking` as the conversation attribute. The `/hotel.xhtml` page begins the natural conversation by loading the current hotel according to the value of the hotelId param. This param is initialized from a request query parameter: `http://seam-booking-example/hotel.seam?hotelId=10`. The `hotelBooking` action then uses this parameter value to initialize the hotel in the conversation context through an `@Factory` method.

Note that all pages other than `/hotel.xhtml` specify `conversation-required=”true”`. This ensures that should a user attempt to access one of these pages outside the context of a long-runningconversation, the user will be redirected to the `no-conversation-view-id` defined in the `<pages>` tag.

Navigation Namespace

Schema URI

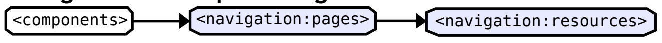
http://jboss.com/products/seam/navigation

Schema XSD

http://jboss.com/products/seam/navigation-2.1.xsd

The navigation namespace provides the ability to externalize configuration of the `Pages` component from the `pages.xml` file and override the location of pages configuration files.

Navigation Namespace Diagram



Hot Tips

Manage your conversations through page navigation. This is generally up to personal preference, but from experience the navigation approach to conversation management tends to lead to more maintainable code. The navigation approach provides clear boundaries for the conversation based on user navigation rather than trying to relate conversation boundaries to page components the user interacts with.

Page Navigation, continued

Navigation Namespace Elements

Element	Description
`<navigation:pages>`	Configures the `Pages` component which drives navigation based on the `/WEB-INF/pages.xml` file.
`<navigation:resources>`	Allows you to specify a list of pages configuration files. Setting this value overrides the default `/WEB-INF/pages.xml`.

Navigation Namespace Example

Below is an example of `<navigation:pages>` definition:

Navigation Namespace Example

```
<components
  xmlns="http://jboss.com/products/seam/components"
  xmlns:navigation=
    "http://jboss.com/products/seam/navigation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/navigation
     http://jboss.com/products/seam/navigation-2.1.xsd
     http://jboss.com/products/seam/components
     http://jboss.com/products/seam/components-2.1.xsd">
  <navigation:pages http-port="8080" https-port="8443"
      no-conversation-view-id="/main.xhtml"
      login-view-id="/home.xhtml" />
  ... ...
</components>
```

The ports have now been defined for both `HTTP` and `HTTPS`. In addition, if a page has been configured as `conversation-required="true"`, the user will be redirected to `/main.xhtml` if a long-running conversation is not in progress. The `login-view-id` specifies that the user should be redirected to `/home.xhtml` if a login is required by a page definition and the user has not yet logged in.

JSF COMPONENT ANNOTATIONS

The definition of components found in a typical JSF application is simple when using Seam. Data models, converters, and validators can be defined quickly with no XML configuration required through use of component annotations.

Datamodel Annotations

The following annotations simplify display of a clickable `<h:dataTable>` in JSF backed by a `DataModel` by directly binding a Collection to action attributes.

Annotation	Use	Description
`@DataModel`	Field, Method	Turns the annotated field or getter method into a JSF `DataModel` object, and implies the `@Out` annotation. Can be used with a List, Map, or Set.
`@DataModelSelection`	Field, Method	Injects the user's `@DataModel` selection. The actual object instance selected is injected.
`@DataModelSelectionIndex`	Field, Method	Injects the row index of the user's `@DataModel` selection.

Conversion and Validation Annotations

The following annotations allow you to quickly define Seam components as custom JSF converters and validators

Annotation	Use	Description
`@Converter`	Type	Allows a Seam component to act as a JSF converter. The annotated class must be a Seam component, and must implement: `javax.faces.convertConverter`
`@Validator`	Type	Allows a Seam component to act as a JSF validator. The annotated class must be a Seam component, and must implement: `javax.faces.validator.Validator`

JSF Component Annotations, continued

JSF COMPONENT TAGS

An extended JSF tag library is defined by Seam to provide control over conversational navigation, simplified dropdowns, bean validation, and component formatting.

Integrating the Seam Taglib

Taglib URI

http://jboss.com/products/seam/taglib

Taglib Declaration

Facelets	`<html xmlns="http://www.w3.org/1999/xhtml" xmlns:s="http://jboss.com/products/seam/taglib">`
JSP	`<%@ taglib uri="http://jboss.com/products/seam/taglib" prefix="s" %>`

Controlling Navigation

Seam provides a set of JSF components for controlling conversation propagation across both GET and POST requests. These components also extend the capabilities of JSF to allow GET requests to trigger actions and make it simple to define default page actions.

Navigation Tags

Tag	Description
`<s:link>`	Link that performs a GET request to invoke an action and allows conversation propagation to be controlled.
`<s:button>`	Button that performs a GET request to invoke an action and allows conversation propagation to be controlled.
`<s:conversationId>`	Adds the conversation ID to a JSF link or button, especially useful with the `<h:outputLink>`.
`<s:conversationPropagation>`	Allows the conversation propagation to be controlled for a JSF command link or command button.
`<s:defaultAction>`	Configures a button (e.g.<h:commandButton>) as the default action when the form is submitted using the enter key

Navigation Examples

The conversation propagation can be controlled from a link through use of the `<s:link>` component. For instance, if you click on the following link, Seam leaves the current conversation when the `main.xhtml` page is loaded, just as a regular HTTP GET request would do.

Navigation Examples

```
<s:link view-id="/main.xhtml" propagation="none"
    value="Back to Main" />
```

We can also trigger an action during an HTTP GET request:

Navigation Examples

```
<s:link view-id="/hotel.xhtml"
  action="#{hotelBooking.selectHotel(hotel.id)}"
  value="Select Hotel" />
```

The `<s:link>` component has richer conversation-management capabilities than the plain JSF `<commandbutton>`, which simply propagates the conversation context between pages.

Hot Tips

Ports change, so make it easy to change them. The `http-port` and `https-port` can be defined in `components.xml` rather than in `pages.xml` to allow use of wild-cards. Wild-cards are defined as `@propertyValue@` and are replaced with values from a `components.properties` file. These ports can (and generally do) change based on environment which makes it useful to break these values out into a properties file for simple substitution.

JSF Component Tags, continued

What if you want to exit, begin, or end a conversation context from a button click? The following example shows a button that exits the current conversation context.

Navigation Examples

```
<h:commandButton action="main"value="Back to Main">
  <s:conversationPropagation type="none"/>
  <s:defaultAction />
</h:commandButton>
```

In addition, by specifying the button is the `<s:defaultAction/>` the user is returned to `/main.xhtml` through a navigation rule should the user submit the form by pressing the enter key.

Hot Tips

Use of Conversation Propagation in Links and Buttons. It is recommended that you limit the use of conversation propagation for links and buttons to simply choosing whether or not to propagate the current conversation. Propagating the conversation is as simple as ensuring the `conversationId` is sent with the request. Propagation of none leaves the current conversation by not passing the `conversationId`. Potential maintenance difficulties can arise when beginning or ending a conversation in a link or button as this does not provide a clear delineation of conversational boundaries.

Dropdown Selection

When developing JSF pages it is commonly required to associate the possible selections of a dropdown component with a list of objects or an enumeration of values. Unfortunately with standard JSF this requires quite a bit of glue code to achieve. Seam makes this simple through the `<s:selectItems>` and `<s:enumItem>` tags. Even further when the `<s:convertEntity>` and the `<s:selectItems>` tags are combined, you can directly associate JPA or Hibernate managed entities by simply binding a dropdown component to an entity association attribute.

Dropdown Tags

Tag	Description
`<s:convertEntity>`	Converts a managed entity to and from its unique identifier. Used for selecting entities in a dropdown component.
`<s:convertEnum>`	Converts an enum to and from its constant representation. Generally used for selecting enums in a dropdown component.
`<s:enumItem>`	Creates a `SelectItem` from an enum value allowing the label to be specified.
`<s:selectItems>`	Creates a `List<SelectItem>` from a List, Set, DataModel or Array. Iterates over the Collection with a var allowing the `itemLabel` and `itemValue` to be defined through EL.

Dropdown Examples

The following example demonstrates directly associating a managed entity from a List of entities. The `CreditCard` class represents types of credit cards and the `@NamedQuery` allows us to load all `CreditCard` types from the data store.

Dropdown Examples

```
@Entity
@NamedQuery(name="loadCreditCardTypes",
  query="select c from CreditCard as c " +
    "order by c.description")
public class CreditCard implements Serializable {
  @Id
  private Long id;
  private String description;
 // ... ...
}
```

JSF Component Tags, continued

The `Booking` class is then be defined with a `@ManyToOne` reference to the `CreditCard` entity.

Dropdown Examples

```
@Entity
public class Booking implements Serializable {
  // ... ...
  @ManyToOne
  @JoinColumn(name="CC_ID")
  private CreditCard creditCard;

  public CreditCard getCreditCard() {
    return creditCard;
  }
  public void setCreditCard(CreditCard creditCard) {
    this.creditCard = creditCard;
  }
  // ... ...
}
```

In order to load the list of credit card types, the `Booking Action` can define an `@Factory` method which initializes the list of `CreditCard` entities in the conversation context.

Dropdown Examples

```
@Name("bookingAction")
@Scope(CONVERSATION)
public class BookingAction implements Serializable {
  // ... ...
  @In private EntityManager entityManager;

  @Factory("creditCardTypes")
  public List<CreditCard> loadCreditCardTypes() {
    return entityManager
      .createNamedQuery("loadCreditCardTypes")
      .getResultList();
  }
  // ... ...
}
```

The method simply uses the named query we defined previously to load the entities from the current `Entity-Manager` instance. Note that our Seam-managed Persistence Context (SMPC) is named `entityManager`. If your SMPC is named something other than `entityManager`, you will have to configure the `EntityConverter` in `components.xml` (see the UI Namespace).

Finally, we can define a JSF `<h:selectOneMenu>` component which simply binds directly to the `creditCard` attribute of the current `booking` instance.

Dropdown Examples

```
<h:selectOneMenu id="creditCard"
    value="#{booking.creditCard}" required="true">
  <s:selectItems noSelectionLabel=""
    var="type" value="#{creditCardTypes}"
    itemLabel="#{type.description}" />
  <s:convertEntity />
</h:selectOneMenu>
```

The `#{creditCardTypes}` are loaded into the conversation context from the `@Factory` method we defined previously. The `<s:selectItems>` component allows us to iterate over the list of `CreditCard` entities and display the description by referencing the `type` variable. The `<s:convertEntity>` tag ensures that the user selection is converted to an entity for association with the `booking` instance.

Simplifying Validation

When using Seam you can define validations directly on your entity beans that behave like JSF validators. These bean validators are provided by the Hibernate Validator framework

JSF Component Tags, continued

(http://validatior.hibernate.org), but with Seam can be triggered as JSF validations.

Validation Tags

Tag	Description
`<s:validate>`	Triggers entity bean validations for the tagged component on a form submission.
`<s:validateAll>`	Triggers entity bean validations for all components embedded within this tag on a form submission.

Validation Examples

The Seam `Booking` example allows a user to enter her credit card number while booking a hotel. Credit card numbers have a common pattern and should be validated on input. The following example demonstrates how we can apply these restrictions using Hibernate Validator annotations:

Validation Examples

```
@Entity
public class Booking {
  ... ...
  @Length(min=16, max=16,
    message="Credit card number must be 16 digits long")
  @Pattern(regex="\\d*",
    message="Credit card number must be numeric")
  private String creditCardNumber;

  public Booking() {}
  ... ...
```

The `@Length` annotation restricts the String length to 16 characters while the `@Pattern` annotation specifies a regular expression restricting the String to digits only. Once these annotations are added to the entity, we can trigger them as validations during the JSF validations phase.
Simply embedding the `<s:validate>` tag within the `<h:inputText>` component ensures the validation is triggered. If an invalid credit card number is input, the user will be presented with the message defined in the annotation.

Validation Examples

```
<div class="entry">
  <div class="label">
    Credit Card #:
  </div>
  <div class="input">
    <h:inputText id="creditCard"
        value="#{booking.creditCardNumber}">
      <s:validate />
    </h:inputText>
    <s:message id="message"
      styleClass="error errors" />
  </div>
</div>
```

Hot Tips

Keep validations DRY by defining them only once. The Hibernate Validator framework allows you to keep your validations DRY (Don't Repeat Yourself) by only defining them once in the entity. Through hooks with Hibernate these validations are enforced at persist-time and with the JSF integration provided by Seam are also enforced in the UI. When placed in entities, validations can be reused by other JSFpages, services, or even other applications avoiding repetitive logic.

Formatting

While validation becomes simple with Seam, standard JSF validation messages are not very flexible. Although you can

JSF Component Tags, continued

assign CSS classes to customize the look of the error message itself, you cannot alter the appearance of the input field that contains the invalid input. The following formatting tags allow you to decorate invalid fields with styles and messages. In addition to validation message formatting Seam provides formatting components for applying labels directly to JSF input fields, optionally rendering HTML `<div>` and `<span>` tags, and optionally rendering page fragments.

Tag	Description
`<s:decorate>`	"Decorate" a JSF input field when validation fails or when `required="true"` is set using a Facelets template.
`<s:label>`	"Decorate" a JSF input field with the label. The label is placed inside the HTML `<label>` tag, and is associated with the nearest JSF input component.
`<s:message>`	"Decorate" a JSF input field with the validation error message associated with that field.
`<s:div>`	Render an HTML `<div>`. Allows the `<div>` to be optionally rendered through the rendered attribute.
`<s:span>`	Render an HTML <span>. Allows the <span> to be optionally rendered through the rendered attribute.
`<s:fragment>`	A non-rendering component useful for optionally rendering its children through the rendered attribute.

Formatting Examples

To use a Seam decorator, you first define how the decorator behaves using special named JSF facets. The `beforeInvalidField` facet defines what to display in front of the invalid field; the `afterInvalidField` facet defines what to display after the invalid field, and the `<s:message>` tag shows the error message for the input field; and the `aroundInvalidField` facet defines a span or div element that encloses the invalid field and the error message. You also can use the aroundField facet to decorate the appearance of valid (or initial) input fields.

Formatting Examples

```
<f:facet name="beforeInvalidField">
  <h:graphicImage styleClass="errorImg"value="error.png"/>
</f:facet>
<f:facet name="afterInvalidField">
  <s:message/>
</f:facet>
<f:facet name="aroundInvalidField">
  <s:span styleClass="error"/>
</f:facet>
```

Now you can simply enclose each input field in a pair of `<s:decorate>` tags as shown below:

Formatting Examples

```
<s:validateAll>
    ... ...
    <s:decorate>
      <h:inputText value="#{booking.creditCardNumber}"/>
    </s:decorate>
    ... ...
</s:validateAll>
```

The UI Namespace

Schema URI

http://jboss.com/products/seam/ui

Schema XSD

http://jboss.com/products/seam/ui-2.1.xsd

As you have seen, the `<s:convertEntity/>` tag provides the ability to convert entities to and from dropdown selections. In most cases this is as simple as defining `<s:convertEntity>` within the dropdown, but there are cases where this needs to be customized. The UI namespace allows you to configure

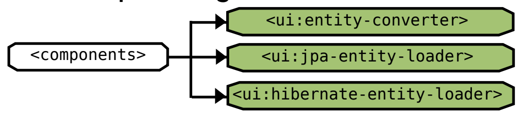
JSF Component Tags, continued

the `EntityConverter` component in the following cases:

- JPA is the persistence provider and your Seam-Managed Persistence Context (SMPC) is named something other than EntityManager
- Hibernate is being used directly as the persistence provider
- You would like to use more than one EntityManager with the EntityConverter

In each of these cases, it is necessary to configure the Entity-Converter component using the UI namespace.

UI Namespace Diagram



UI Namespace Elements

Tag	Description
`<ui:entity-converter>`	Allows a custom `EntityConverter` to be defined which is useful if more than one `EntityManager` is being used.
`<ui:jpa-entity-loader>`	Configures a Seam-Managed Persistence Context (SMPC) named something other than `entityManager`.
`<ui:hibernate-entity-loader>`	Allows a Managed Hibernate Session to be configured for use with the `EntityConverter`. By default assumes the component name `session`.

JSF Component Tags, continued

UI Namespace Examples

The following example configures the `EntityConverter` with a custom `EntityManager` name:

UI Namespace Examples

```
<components xmlns=
    "http://jboss.com/products/seam/components"
  xmlns:persistence=
    "http://jboss.com/products/seam/persistence"
  xmlns:ui="http://jboss.com/products/seam/ui"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/persistence
     http://jboss.com/products/seam/persistence-2.1.xsd
     http://jboss.com/products/seam/ui
     http://jboss.com/products/seam/ui-2.1.xsd
     http://jboss.com/products/seam/components
     http://jboss.com/products/seam/components-2.1.xsd">
  <persistence:managed-persistence-context name="em"
     auto-create="true"persistence-unit-jndi-name=
       "java:/EMFactories/bookingEntityManagerFactory"/>
  <ui:jpa-entity-loader entity-manager="#{em}" />
  ... ...
</components>
```

As you can see, the Seam-managed Persistence Context is given the name `em`. This is simply referenced through an EL expression in the `jpa-entity-loader` configuration.

ABOUT THE AUTHOR

Jacob Orshalick is an independent software consultant, open source enthusiast, and the owner of Focus IT Solutions, an independent software consulting firm. He has a Masters degree in Software Engineering from The University of Texas at Dallas and has eight years of development experience in the retail, financial, and telecommunications industries. You can also find Jacob writing about Seam in his blog.

Blog: http://solutionsfit.com/blog/

Projects: Committer to Seam Framework

RECOMMENDED BOOK

The Seam Framework has simplified Java enterprise Web development forthousands of developers and significantly influenced the broader Java Enterprise Edition platform. Now, the authors of the leading guide to Seam development have systematically updated it to reflect the major improvements and new features introduced with Seam 2.x. The book also introduces Web Beans (JSR-299), the future core of Seam that will transform Java EE Web development.

BUY NOW

books.dzone.com/books/seam-framework

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-46-6
ISBN-10: 1-934238-46-7

50795

9 781934 238455

$7.95