DZone Refcardz

Brought to you by... JBoss by Red Hat

CONTENTS INCLUDE:

- Bijection in a Nutshell
- Contextual Components
- Common Application Configuration
- Seam Security
- Application Framework
- Hot Tips and More...

Core Seam

By Jacob Orshalick

ABOUT SEAM

Seam is a next generation web framework that integrates standard Java EE technologies with a wide variety of nonstandard technologies into a consistent, unified, programming model. Seam drove the development of the Web Beans specification (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 quick reference a handy guide for understanding core concepts, configuration, and tool usage. This quick reference is not intended to cover all of what Seam provides, but will cover the most commonly used annotations and XML elements as of Seam 2.1. In addition, this guide will point you to examples distributed with Seam to see real examples of how the configuration options can be used in practice.

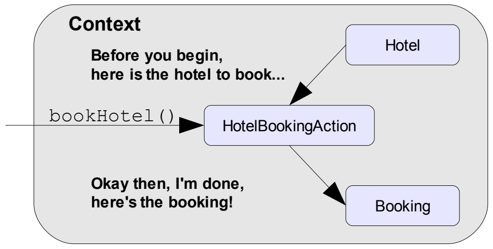
BIJECTION IN A NUTSHELL

Dependency injection is an *inversion of control* technique that forms the core of modern-day frameworks. Traditionally objects have held the responsibility for obtaining references to the objects they collaborate with. These objects are extroverted as they reach out to get their dependencies. This leads to tight coupling and hard to test code.

Dependency injection allows us to create introverted objects. The objects dependencies are injected by a container or by some external object (e.g. a test class). Bijection is described by the following formula:

dependency injection + context = bijection

With bijection, when dependencies are injected context counts! Dependencies are injected prior to each component method invocation. In addition, components can contribute to the context by outjecting values.


Context
Before you begin, here is the hotel to book...
Hotel
bookHotel()
HotelBookingAction
Okay then, I'm done, here's the booking!
Booking

As you can see the `HotelBookingAction` is scoped to and executes within a context. This behavior allows us to quit worrying about shuffling values into and out of contexts like the `HttpSession`, allows components to hold state, and unifies the component model across application tiers.

CONTEXTUAL COMPONENTS

When a method is invoked on a component, its dependencies are injected from the current context. Seam performs a context lookup in the following order of precedence: *Event Scope, Page Scope, Conversation Scope, Session Scope, Business Scope, Application Scope.*

Component Annotations

Component Definition Annotations

In order for your Seam components to take advantage of bijection, you must register them with the Seam container. Registering your component with Seam is as simple as annotating it with `@Name`. The following annotations will register your component and define its lifecycle.

Annotation	Use	Description
`@Name`	Type	Declares a Seam component by name. The component is registered with Seam and can be referenced by name through Expression Language (EL), injection, or a context lookup.
`@Scope`	Type	Defines the scope (or context) the Seam component will be placed into by the container when created.
`@AutoCreate`	Type	Specifies that a component should be created when being injected if not available in the current context.
`@Startup`	Type	Indicates that an application scoped component should be created when the application is initialized or that a session component should be created when a session is started. Not valid for any other contexts.
`@Install`	Type	Declares whether a Seam component should be installed based on availability of dependencies or precedence.
`@Role`	Type	Defines an additional name and scope associated with the component. The `@Roles` annotation allows definition of multiple roles.

Contextual Components, continued

Component Bijection Annotations: Once you have defined a component, you can specify the dependencies of your component and what the component contributes back to the context.

Annotation	Use	Description
`@In`	Field, Method	Declares a dependency that will be injected from the context, according to context precedence, prior to a method invocation. Note that these attributes will be disinjected (or set to null) after the invocation completes.
`@Out`	Field, Method	Declares a value that will be outjected after a method invocation to the context of the component (implicit) or a specified context (explicit).

Component Lifecycle Annotations: The following annotations allow you to control the lifecycle of a component either by reacting to events or wrapping the component entirely.

Annotation	Use	Description
`@Create`	Method	Declares that a method should be called after component instantiation.
`@Destroy`	Method	Declares that a method should be called just before component destruction.
`@Factory`	Method	Marks a method as a factory method for a context variable. A factory method is called whenever no value is bound to the named context variable and either initializes and outjects the value or simply returns the value to the context.
`@Unwrap`	Method	Declares that the object returned by the annotated getter method is to be injected instead of the component itself. Referred to as the "manager component" pattern.

Component Events Annotations: Through Seam's event model, components can raise events or listen for events raised by other components through simple annotation. In addition, Seam defines a number of built-in events that the application can use to perform special kinds of framework integration (see http://seamframework.org/Documentation, Contextual Events).

Annotation	Use	Description
`@RaiseEvent`	Method	Declares that a named event should be raised after the method returns a non-null result without exception.
`@Observer`	Method	Declares that a method should be invoked on occurrence of a named event or multiple named events.

The Components Namespace

Schema URI

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

Schema XSD

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

So far we've seen how components can be declared using annotations. In most cases this is the preferred approach, but there are some situations when component definition through annotations is not an option:

- when a class from a library outside of your control is to be exposed as a component
- when the same class is being configured as multiple components

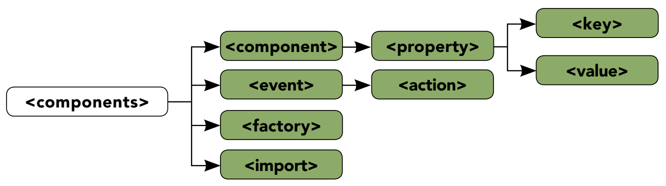
In addition, you may want to configure values into a component that could change by environment, e.g. ip-addresses, ports, etc. In any of these cases, we can use XML to configure the component through the components namespace.

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 none
	→	Containment

Components Namespace Diagram



Components Namespace Elements

Element	Description
`<component>`	Defines a component in the Seam container.
`<event>`	Specifies an event type and the actions to execute on occurrence of the event.
`<factory>`	Lets you specify a value or method binding expression that will be evaluated to initialize the value of a context variable when it is first referenced. Generally used for aliasing.
`<import>`	Specifies component namespaces that should be imported globally which allows referencing by unqualified component names. Specified by package.
`<action>`	Specifies an action to execute through a method binding expression.
`<property>`	Injects a value or reference into a Seam component. Can use a value or method binding expression to inject components.
`<key>`	Defines the key for the following value when defining a map.
`<value>`	For list values, the value to be added. For map values, the value for the preceding key.

Component Element Attributes: The attributes of the component element are synonymous with component definition through annotations.

Element	Description
`name`	Declares a component by name; synonymous with `@Name`.
`class`	References the Java class of the component implementation.
`scope`	Defines the scope (or context) the Seam component will be placed into by the container when created.
`precedence`	Precedence is used when a name-clash is found between two components (higher precedence wins).
`installed`	Boolean indicating whether the component should be installed.
`auto-create`	Specifies that a component should be created when being injected if not available in the current context.
`startup`	Indicates that an application scoped component should be created when the application is initialized or that a session component should be created when a session is started. Not valid for any other contexts.
`startupDepends`	A list of other application scope components that should be started before this one, if they are installed.
`jndi-name`	EJB components only. The JNDI name used to lookup the component. Only used with EJB components that don't follow the global JNDI pattern.

Components Namespace Example

The following examples demonstrate how component configuration is possible with Seam. The `hotelBooking` component is configured for injection of a `paymentService` instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<components
     xmlns="http://jboss.com/products/seam/components"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation=
        "http://jboss.com/products/seam/components
        http://jboss.com/products/seam/components-2.1.xsd">

    <component name="hotelBooking">
     <property name="paymentService">
        #{paymentService}
     </property>

    </component>
    <component name="paymentService" scope="APPLICATION"
           class="com.othercompany.PaymentServiceClient">
     <property name="ipAddress">127.0.0.1</property>
     <property name="port">9998</property>

    </component>
</components>
```

Note that we specified a name and class for the `PaymentServiceClient` instances. This is required as the `PaymentServiceClient` is a library implementation. The name is always required, but the class can be omitted if you are simply configuring the values of a component with an `@Name` annotation as shown with the `hotelBooking`. The scope is restricted to the scopes provided by Seam; here we use `APPLICATION`.

Simplify your component configuration through use of namespaces.

Seam makes this quite simple through use of the `@Namespace` annotation. Just create a file named package-info.java in the package where your components live:

```
@Namespace(value="http://solutionsfit.com/example/booking")
package com.solutionsfit.example.booking;
import org.jboss.seam.annotations.Namespace;
```

Now we can reference the namespace in our `components.xml`:

```
<components xmlns=
  "http://jboss.com/products/seam/components"
  xmlns:payment="http://solutionsfit.com/example/booking">
  ... ...
  <payment:hotel-booking
     payment-service="#{paymentService}">
```

Note that component names and attribute names are specified in hyphenated form when using namespaces. To gain the benefits of auto-completion and validation, a schema can also be created to represent your components. A custom schema can import the components namespace to reuse the defined component types.

RAPID APPLICATION DEVELOPMENT

Seam-gen is a rapid application generator shipped with the Seam distribution. With a few command line commands, seam-gen generates an array of artifacts for Seam projects to help you get started. In particular, seam-gen is useful for the following.

- Automatically generate an empty Seam project with common configuration files, a build script, and directories for Java code and JSF view pages.
- Automatically generate complete Eclipse and NetBeans project files for the Seam project.
- Reverse-engineer entity bean objects from relational database tables.
- Generate template files for common Seam components.

Seam-gen Commands

Seam-gen command should be executed in the directory of your Seam distribution. To get started execute the `seam setup` command. Each command provides useful prompts that will guide you through the configuration specifics.

Command	Description
`seam setup`	Configures seam-gen for your environment: JBoss AS installation directory, Eclipse workspace, and database connection.
`seam new-project`	Creates a new deployable Seam project based on the configuration settings provided in the setup.
`seam -D[profile] deploy`	Deploys the new project you've created with the configuration specific to the [profile] specified, i.e. dev, test, or prod.
`seam new-action`	Creates a simple web page with a stateless action method. Generates a facelets page and component for your project.
`seam new-form`	Creates a form with an action.
`seam generate-entities`	Generates an application from an existing database. Simply ensure that your setup points to the appropriate database.
`seam generate-ui`	Generates an application from existing JPA/EJB3 entities placed into the src/model folder.
`seam restart`	Restarts the application on the server instance.

Seam-gen Source Directories

The source directories created by seam-gen are each dedicated to holding certain types of source files. This enables seam-gen to determine what tests need to be executed as well as selectively hot-deploy your Seam components.

`/src/main`	Classes to include in the main (static) classpath.
`/src/hot`	Classes to include in the hot deployable classpath when running in development mode (see the Hot Tip below).
`/src/test`	Test classes should be placed in this source folder. These classes will not be deployed but will be executed as part of the test target.

Seam-gen Profiles

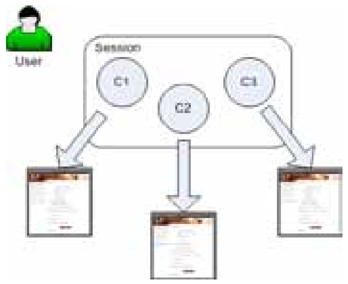
Seam-gen supports the concept of "profile". The idea is that the application probably needs different settings (e.g. database settings, etc) in the development, test, and production phases. So, the project provides alternative configuration files for each of the three scenarios. In the resources directory, you will find the following configuration files:

`/META-INF/persistence-*.xml`	Provides configuration of a JPA persistence-unit for each environment.
`import-*.sql`	Imports data into a generated schema based on deployment environment. Generally useful for testing purposes.
`components-*.properties`	Allows wild-card replacement of Seam component properties based on environment (see Hot Tip in the Core Namespace for more information).

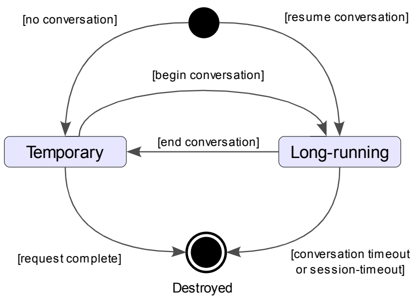
CONVERSATION MANAGEMENT

The Conversation Model in a Nutshell

The conversation model is the core of Seam. It provides the basis of context management in Seam applications. A session holds potentially many concurrent conversations with a user, each tied to its own unit of work with its own context.



On every user request a temporary conversation is created or a long-running conversation is resumed. A conversation is resumed when a valid conversation-id `(cid)` is sent with the request.



* *Note that the filled circle indicates the initial state, while the hollow circle containing a smaller filled circle indicates an end state.*

Temporary Conversations	A temporary conversation is started on every request unless an existing long-running conversation is being resumed. Even if your application does not explicitly specify conversation handling a conversation will be initialized for the request. A temporary conversation survives a redirect.
Long-running Conversations	A temporary conversation is promoted to longrunning if you tell Seam to begin a long-running conversation. Promoting a conversation to long-running informs Seam to maintain the conversation between requests. The conversation is demoted to temporary if you tell Seam to end a long-running conversation.

Conversation Management Annotations

These annotations provide declarative conversation management through Seam components.

Annotation	Use	Description
`@Conversational`	Type, Method	A component or method annotated as `@Conversational` can only be accessed in the context of a long-running conversation.
`@Begin`	Method	Marks a method as beginning a long-running conversation. The long-running conversation will only end upon method return if: the method is void return-type or the method returns a non-null outcome.
`@End`	Method	Marks a method as ending a long-running conversation. The long-running conversation will only begin upon method return if: the method is void return-type or the method returns a non-null outcome.

Hot Tip

Don't restart your application, use incremental hot deployment. Seam-gen provides support for exploded deployment of your application. Exploded deployment allows changes to any XHTML file or the `pages.xml` file to be redeployed without an application restart. In order to enable incremental hot deployment, you must set Seam and Facelets into debug mode. By default, a seamgen'd application is debug enabled in the dev profile. If you are using JavaBean action components, they are also deployable without an application restart. Just ensure they are placed in your `/src/hot` folder and debug mode is turned on.

Conversation Management, continued

Conversation Management Usage

Some example usage patterns for conversation propagation.

1. Begin a conversation when the `selectHotel` method is invoked:

```
@Begin
public void selectHotel(Hotel selectedHotel)
{
  hotel = em.merge(selectedHotel);
}
```

2. End a conversation when the `booking` is confirmed:

```
@End(beforeRedirect="true")
public void confirm()
{
  em.persist(booking);
  ...
```

Note the use of the `beforeRedirect` parameter. Recall that a conversation is really only demoted to temporary when ended and survives a redirect. Here the conversation is destroyed before the redirect.

3. Begin a conversation with manual flushing enabled:

```
@In private EntityManager em;

@Begin(flushMode=FlushModeType.MANUAL)
public void editBooking(int selectedId)
{
   booking = em.find(Booking.class, selectedId);
}
```

This ensures that the booking would not be persisted until an `EntityManager.flush()` is invoked manually. This is only usable with an SMPC and Hibernate as the persistence provider.

4. Begin or join the conversation when the hotel page is accessed:

```
<page view-id="/hotel.xhtml"
      login-required="true">
      <description>
      View hotel: #{hotel.name}
      </description>
      <begin-conversation join="true">
```

This example demonstrates conversation management through the `pages.xml` file. The `pages.xml` file provides a navigation-centric approach to conversation management directly relating the boundaries of a conversation to page navigation.

5. Leave the scope of an existing conversation to return to the main page:

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

Use of the `<s:link>` and `<s:button>` tags in your view are common to either propagate a conversation across a GET request or to leave the scope of an existing conversation.

Seam Distribution Conversation Management Examples

The following examples are distributed with Seam and provide great resources for configuring your application.

Example	Directory	Description
Seam Booking	`examples/booking`	Demonstrates the most basic conversation management through use of annotations.
Nested Booking	`examples/nestedbooking`	Demonstrates the use of nested conversations through annotations.
Seam Space	`examples/seamspace`	Demonstrates conversation management through pages.xml as well as use of natural conversations.

Hot Tip

Let your conversations time-out!

It is a common misconception that conversations should always be ended when navigating elsewhere in an application. Seam ensures that these abandoned, or background conversations are cleaned up after a period of time (the `conversation-timeout` setting). The conversation that the user last interacted, the foreground conversation, will only timeout when the session times out. See the Core Namespace to learn how to configure the `conversation-timeout` setting.

COMMON APPLICATION CONFIGURATION

The Core Namespace

Schema URI

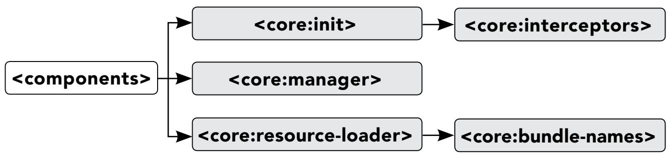
http://jboss.com/products/seam/core

Schema XSD

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

The core namespace includes support for configuration of Seam components found in org.jboss.seam.core. This includes the JNDI pattern used to lookup EJB components, debug mode settings, conversation management, inclusion of custom resource bundles, and POJO cache settings.

Core Namespace Diagram



Core Namespace Elements

Element	Description
`<core:init>`	Initialization parameters including debug settings and configuration of the `jndiPattern` for EJB users.
`<core:resource-loader>`	Allows configuration of custom resource bundle-names for loading messages.
`<core:bundle-names>`	Custom bundle names can be specified which are searched depth-first for messages. This overrides the standard `messages.properties`.
`<core:manager>`	Configures conversation management settings such as timeouts, parameter names, uri-encoding, and the default-flush-mode.
`<core:interceptors>`	Configures the list of interceptors that should be enabled for all components. All built-in interceptors must be specified with any additional interceptors.

Core Namespace Example

The following example demonstrates use of the core namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<components
   xmlns="http://jboss.com/products/seam/components"
   xmlns:core="http://jboss.com/products/seam/core"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
     " http://jboss.com/products/seam/core
      http://jboss.com/products/seam/core-2.1.xsd
      http://jboss.com/products/seam/components
      http://jboss.com/products/seam/components-2.1.xsd">

   <core:init jndi-pattern="@jndiPattern@" debug="@debug@"/>

   <core:manager conversation-timeout="600000"
                 conversation-id-parameter="cid"
                 default-flush-mode="MANUAL" />
   ... ...
```

The above configuration tells Seam how to find EJB components in the JNDI registry and whether to operate in `debug` mode allowing features such as incremental deployment. The `manager` configuration ensures that background conversations timeout after 10 minutes through the `conversation-timeout` setting. Note the setting is configured in milliseconds. The cid parameter simply specifies how our conversation ID should be represented in a query string, e.g. `/book.seam?cid=20`.

Common Application Configuration, continued

Note the use of the `default-flush-mode` setting. If you are using a Seam-managed Persistence Context (SMPC), which will be discussed shortly, this setting will save constant repetition of the flush-mode setting when beginning conversations.

Hot Tip It is generally a good idea to keep some application configuration in properties files. This makes it simple for administrators to easily tweak deployment-specific settings (e.g. database settings).

Seam lets you place wildcards of the form @propertyName@ in your components.xml file. Seam uses the components.properties file in the classpath to replace the properties by name. Seam-gen projects use this approach by default to dynamically replace the jndiPattern and debug settings.

The Transaction Namespace

Transaction Namespace URI

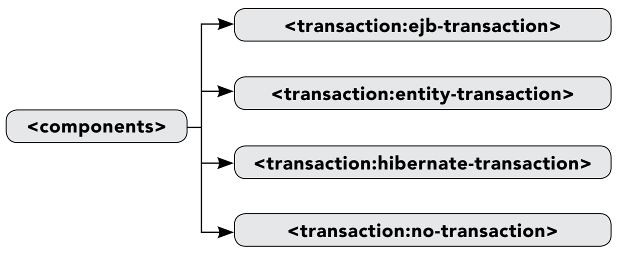
http://jboss.com/products/seam/transaction

Transaction Namespace XSD

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

Transactions are an essential feature for database-driven web applications. In each conversation, we typically need to update multiple database tables. If an error occurs in the database operation (e.g. a database server crashes), the application needs to inform the user, and all the updates this conversation has written into the database must be rolled back to avoid partially updated records. In other words, all database updates in the conversation must happen inside an atomic operation. Seam-managed transactions along with Seam-managed persistence enables you to do just that.

Transaction Namespace Diagram



Transaction Namespace Elements

Element	Description
`<transaction: ejb-transaction>`	Configures the EJB transaction synchronization component which should be installed if you are working in a Java EE 5 environment.
`<transaction: entity-transaction>`	Configures JPA `RESOURCE_LOCAL` transaction management where the the `managedpersistence-context` is injected as an Expression Language (EL) attribute. This attribute is not required if your `EntityManager` is named `entityManager`.
`<transaction: hibernate-transaction>`	Configures Hibernate managed transactions where the `persistence:managed-hibernatesession` is injected as an Expression Language (EL) attribute. This attribute is not required if your `Session` is named `session`.
`<transaction: no-transaction>`	Disables Seam-managed transactions.

The Transaction Namespace, continued

Transaction Namespace Example

The following example configures the EJB transaction synchronization component:

```
<components
   xmlns="http://jboss.com/products/seam/components"
   xmlns:persistence=
      "http://jboss.com/products/seam/persistence"
   xmlns:transaction=
      "http://jboss.com/products/seam/transaction"
   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/transaction
        http://jboss.com/products/seam/transaction-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"/>
   <transaction:ejb-transaction/>
... ...
```

Note that this indicates the application is being deployed to a JEE 5 environment.

The Persistence Namespace

Persistence Namespace URI

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

Persistence Namespace XSD

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

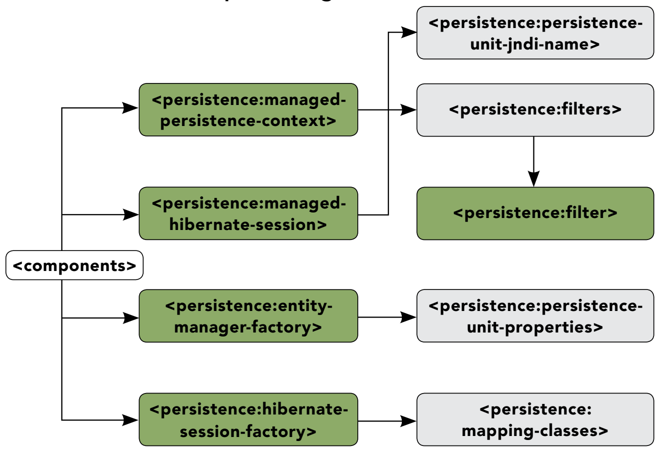
When JPA is used within an EJB environment, an extended persistence context can be associated with a stateful session bean. An extended persistence context lives between requests and is scoped to the stateful component allowing entities to remain managed. The EJB3 approach has several shortcomings:

- What if my application does not operate in an EJB environment?
- It can be tricky to ensure that the persistence context is shared between loosely coupled components.
- EJB3 does not include the concept of manual flushing.

A Seam-managed persistence context (or SMPC) is available for use in both environments and alleviates these issues altogether. An SMPC is just a built-in Seam component that manages an instance of `EntityManager` or Hibernate `Session` in the conversation context. You can inject it with @In.

Note that many of the options for configuring a `HibernateSessionFactory` have been omitted due to space constraints. For more options on configuring your `HibernateSessionFactory`, see the Seam Reference Documentation (http://seamframework.org/Documentation).

Persistence Namespace Diagram



→

The Persistence Namespace, continued

Persistence Namespace Elements

Element	Description
`<persistence:entity-manager-factory>`	Informs Seam to bootstrap a JPA `EntityManagerFactory` from your `persistence.xml` file.
`<persistence:managed-persistence-context>`	Configures a Seam managed JPA `EntityManager` to be available via injection.
`<persistence:persistence-unit-jndi-name>`	Allows the persistence unit's JNDI name to be embedded as an element rather than an attribute.
`<persistence:hibernate-session-factory>`	If you choose to use Hibernate directly, informs Seam to bootstrap a `HibernateSessionFactory`. Allows your Hibernate configuration to be specified through a variety of approaches with the sub-elements shown in the namespace diagram.
`<persistence:managed-hibernate-session>`	Configures a Seam-managed Hibernate `Session` to be available via injection.
`<persistence:filters>`	Defines a set of Hibernate filters. Can only be used with Hibernate.
`<persistence:filter>`	Defines a Hibernate filter that is enabled whenever an `EntityManager` or Hibernate `Session` is first created. Can only be used with Hibernate.
`<persistence:name>`	Provides the name of the Hibernate filter.
`<persistence:parameters>`	Specifies key-value pairs setting the parameter values as Expression Language (EL) expressions for the Hibernate filter.

Persistence Namespace Example

The following example demonstrates use of the persistence namespace:

```
<components
   xmlns="http://jboss.com/products/seam/components"
   xmlns:persistence=
      "http://jboss.com/products/seam/persistence"
   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/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"/>
... ...
</components>
```

As configured above, a Seam-managed `EntityManager` can be injected as:

```
@In EntityManager em;
```

The `EntityManagerFactory` is looked up by the configured JNDI name so this name must be consistent with your `persistence.xml` configuration. If you are using JBoss this can be configured with the following in the properties of your persistence-unit definition:

```
<property name="jboss.entity.manager.factory.jndi.name"
   value="java:/EMFactories/bookingEntityManagerFactory"/>
```

Seam Distribution Persistence Configuration Examples

The following examples are distributed with Seam and provide great resources for configuring your application.

Example	Directory	Description
Booking	`examples/booking`	Demonstrates configuration of using a basic EJB PersistenceContext managed by the EJB container.
Hibernate Booking	`examples/hibernate`	Demonstrates configuration of using direct Hibernate persistence with a Seam-managed Hibernate `Session`.
JPA Booking	`examples/jpa`	Demonstrates use of an SMPC with JPA `RESOURCE_LOCAL` transaction management usable in non-EJB containers like Tomcat.

SEAM SECURITY

Security is arguably one of the most important aspects of application development. Unfortunately due to its complexity security is often an afterthought which can lead to gaping holes in an application for malicious users to take advantage of. Fortunately, when using Seam this complexity is hidden making it simple to ensure that your next application is secure.

Seam Security, continued

Seam security provides extensive configuration options making it useful in a wide array of implementations but also making it worthy of its own reference card. The basics of Seam security will be covered here which will address most use-cases and will allow you to get up-and-running quickly.

The Security Namespace

Security Namespace URI

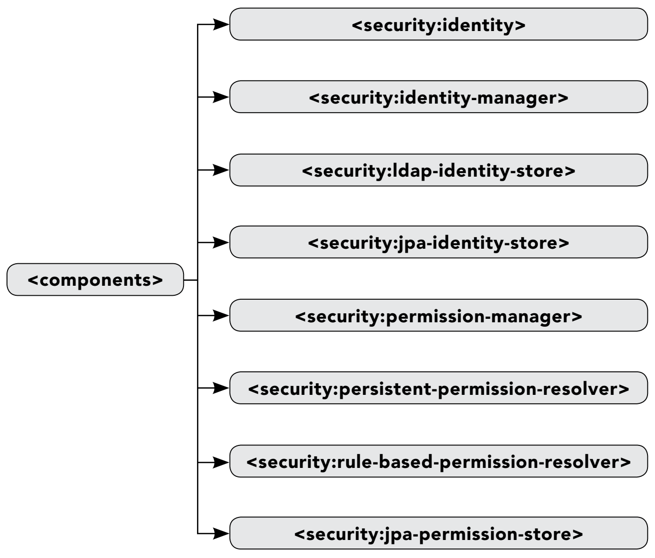
http://jboss.com/products/seam/security

Security Namespace XSD

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

The security namespace provides the hooks necessary to customize Seam security to fit your application needs whether you intend to use LDAP, a relational database, or any other custom authentication / authorization scheme.

Security Namespace Diagram



Security Namespace Elements

Element	Description
`<security:identity>`	Configures the application-specific authentication implementation for the Seam `Identity` component. The `IdentityManager` is used for authentication by default.
`<security:identity-manager>`	Configures the management of a Seam application's users and roles for a type of identity store (database, LDAP, etc). The `JpaIdentityStore` is the default.
`<security:ldap-identity-store>`	Declares an identity store that allows users and roles to be stored inside an LDAP directory.
`<security:jpa-identity-store>`	Declares an identity store that allows users and roles to be stored inside a relational database.
`<security:permission-manager>`	Configures a `PermissionManager` instance for management of persistent permissions which requires a `PermissionStore` instance.
`<security:persistent-permission-resolver>`	Allows configuration of a custom permission store. A permission store can be implemented by implementing the `PermissionStore` interface.
`<security:rule-based-permission-resolver>`	Allows the Drools rule base name to be specified when rule-based permissions are in use.
`<security:jpa-permission-store>`	`PermissionStore` implementation allowing user and role permissions to be stored and retrieved as JPA entities.

Seam Security, continued

Security Namespace Example

The `JpaIdentityStore` component provided by Seam 2.1 makes it simple to store users and roles inside a relational database.

```
<components
   xmlns="http://jboss.com/products/seam/components"
   xmlns:drools="http://jboss.com/products/seam/drools"
   xmlns:security="http://jboss.com/products/seam/security"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://jboss.com/products/seam/security
       http://jboss.com/products/seam/security-2.1.xsd
       http://jboss.com/products/seam/drools
       http://jboss.com/products/seam/drools-2.0.xsd
       http://jboss.com/products/seam/components
       http://jboss.com/products/seam/components-2.1.xsd">
<security:jpa-identity-store
user-class="org.jboss.seam.example.booking.MemberAccount"
role-class="org.jboss.seam.example.booking.MemberRole" />
... ...
```

Above we need not configure the `IdentityManager` as the `JpaIdentityStore` is assumed by default.

Securing your Application

Seam makes it easy to declare access constraints on web pages, UI components, and Java methods via tags, annotations, and JSF Expression Language (EL) expressions. Through this declarative approach Seam provides you with the ability to easily achieve a layered approach to security.

Security Annotations

Annotation	Use	Description
`@Restrict`	Type, Method	Allows a component to be secured either at the method or the class level through evaluation of an Expression Language (EL) expression.
`@Read`	Method, Parameter	Declares a type-safe permission check to determine whether the current user has read permission for the specified class.
`@Update`	Method, Parameter	Declares a type-safe permission check to determine whether the current user has update permission for the specified class.
`@Insert`	Method, Parameter	Declares a type-safe permission check to determine whether the current user has insert permission for the specified class.
`@Delete`	Method, Parameter	Declares a type-safe permission check to determine whether the current user has delete permission for the specified class.
`@PermissionCheck`	Annotation	Declares a custom security annotation for performing type-safe permission checks where the required permission is the lower-case representation of the annotation name.
`@RoleCheck`	Annotation	Declares a custom security annotation for performing type-safe role checks where the required role is the lower-case representation of the annotation name.

Security Expression Language (EL) Functions

These functions provide convenience for invoking role or permission checks through Expression Language (EL). This can be useful in your Facelets pages, in pages.xml, or in using the @Restrict annotation to provide security restrictions.

Function	Description
`#{s:hasRole(roleName)}`	Expression Language (EL) function that invokes the `hasRole` method on the `Identity` component. Returns true if the current user has the specified role.
`#{s:hasPermission(target, action)}`	Expression Language (EL) function that invokes the `hasPermission` method on the Identity component. Delegates the call to the configured `PermissionResolver`.

Seam Distribution Security Examples

The following examples are distributed with Seam and provide great resources for configuring your application.

Example	Directory	Description
Seam Booking	`examples/booking`	Demonstrates the most basic authentication through use of an authenticate-method.
Seam Space	`examples/seamspace`	Demonstrates use of rule-based-permission-resolver, jpa-identity-store, and jpa-permission-store.

APPLICATION FRAMEWORK

The Seam CRUD application framework essentially provides prepackaged Data Access Objects (DAOs). DAOs are highly repetitive as the DAOs for each entity class are largely the same. They are ideal for code reuse. Seam provides an application framework with built-in generic DAO components. You can develop simple CRUD web applications in Seam without writing a single line of Java "business logic" code.

The Framework Namespace

Framework Namespace URI

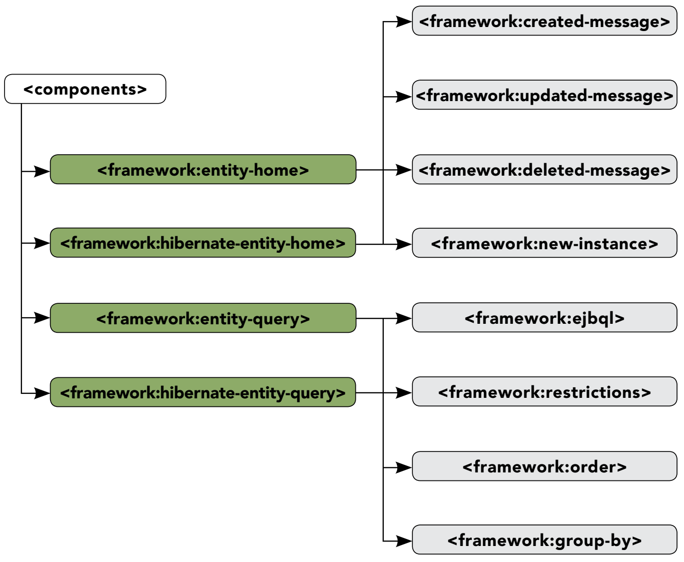
http://jboss.com/products/seam/framework

Framework Namespace XSD

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

The framework namespace configures components found in the `org.jboss.seam.framework` package.

Framework Namespace Diagram



Framework Namespace Elements

Element	Description
`<framework:entity-query>`	Configures an `EntityQuery` controller instance for executing JPA queries.
`<framework:entity-home>`	Configures an `EntityHome` controller instance for managing JPA entities.
`<framework:hibernate-entity-query>`	Configures a `HibernateQuery` controller instance for executing Hibernate queries.
`<framework:hibernate-entity-home>`	Configures a `HibernateEntityHome` controller instance for managing Hibernate entities.
`<framework:new-instance>`	Declares the new instance managed by a home controller.
`<framework:ejbql>`	Declares the new instance managed by a home controller.
`<framework:order>`	Defines the order-by property for a `Query` instance.
`<framework:restrictions>`	Defines the where clause of a `Query` instance.
`<framework:group-by>`	Defines the group-by clause for a `Query` instance.
`<framework:created-message>`	A status message added when a `Home` controller creates a new entity instance.
`<framework:updated-message>`	A status message added when the `Home` controller updates an entity instance.
`<framework:deleted-message>`	A status message added when the `Home` controller deletes an entity instance.

Application Framework, continued

Framework Namespace Example

In the booking example we could use an `EntityHome` instance to manage retrieval of the Hotel entity. The configuration below achieves this:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns=
      "http://jboss.com/products/seam/components"
   xmlns:persistence=
      "http://jboss.com/products/seam/persistence"
   xmlns:framework="http://jboss.com/products/seam/framework"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://jboss.com/product s/seam/framework
       http://jboss.com/products/seam/framework-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"/>
<factory name="hotel"
   value="#{hotelHome.instance}"
   scope="stateless"
   auto-create="true"/>
```

Framework Namespace Example, continued

```
<framework:entity-home name="hotelHome"
      entity-class="com.jboss.seam.booking.Hotel"
      scope="conversation" entity-manager="#{em}"
      auto-create="true">
   <framework:id>#{hotelId}</framework:id>
</framework:entity-home>
... ...
```

When a `hotel` is requested for injection or through Expression Language (EL) the `hotelHome` will be invoked to retrieve the hotel based on the current `hotelId`. This of course requires that the requested `hotelId` be made available in the current context. This can easily be achieved through use of a request parameter or outjection.

Seam Distribution Framework Examples

The following examples are distributed with Seam and provide great resources for configuring your application.

Example	Directory	Description
Seam Pay	`examples/seampay`	Demonstrates use of entity-query and entityhome elements.
DVD Store	`examples/dvdstore`	Demonstrates use of entity-query and entityhome elements.
Contact List	`examples/contactlist`	Demonstrates use of entity-query and entityhome with embedded namespace elements. Also demonstrates use of restrictions.

ABOUT THE AUTHOR

Jacob Orshalick

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.

Publications

- *Seam Framework: Experience the Evolution of Java EE*

Blog

http://solutionsfit.com/blog/

Projects

Committer to 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: 978-1-934238-31-8
ISBN-10: 1-934238-31-7
50795
9 781934 238318
$7.95

Version 1.0