## Seam Servlet Module

# Reference Guide

**Lincoln Baxter III**

**Nicklas Karlsson**

**Dan Allen**

Introduction ..... v
**1. Installation** ..... 1
**2. Servlet event propagation** ..... 3
2.1. Servlet context lifecycle listener ..... 3
2.2. Application initialization ..... 3
2.3. Servlet request lifecycle listener ..... 4
2.4. Servlet response lifecycle listener ..... 5
2.5. Session lifecycle listener ..... 6
2.6. Session activation listener ..... 7
**3. Injectable Servlet objects and request state** ..... 9
3.1. @Inject @RequestParam ..... 9
3.2. @Inject ServletContext ..... 10
3.3. @Inject HttpSession ..... 10
3.4. @Inject HttpServletRequest ..... 11
3.5. @Inject @ContextPath ..... 11
3.6. @Inject List<Cookie> ..... 11
**4. Servlet Context attribute BeanManager provider** ..... 13

## Introduction

The goal of the Seam Servlet module is to provide portable enhancements to the Servlet API. Features include producers for implicit Servlet objects and HTTP request state, propagating Servlet events to the CDI event bus, forwarding uncaught exceptions to the Seam Catch handler chain (planned) and binding the BeanManager to a Servlet context attribute for convenient access.

# Installation

Most features of Seam Servlet are installed automatically by including the seam-servlet.jar and seam-servlet-api.jar in the web application library folder. If you are using *Maven* [http://maven.apache.org/] as your build tool, you can add the following single dependency to your pom.xml file:

```
<dependency>
   <groupId>org.jboss.seam.servlet</groupId>
   <artifactId>seam-servlet-impl</artifactId>
   <version>${seam.servlet.version}</version>
</dependency>
```

**Tip**

Substitute the expression ${seam.servlet.version} with the most recent or appropriate version of Seam Servlet. Alternatively, you can assign the version number to the property of the same name inside the `properties` element of your pom.xml.

In a Servlet 3.0 or Java EE 6 environment, *your configuration is now complete*.

If you are using Servlet 2.5 or Java EE 5, then you need to manually register several Servlet components in your application's web.xml to activate the features provided by this module:

```
<listener>
  <listener-class>org.jboss.seam.servlet.event.ServletEventBridgeListener</listener-class>
</listener>

<filter>
  <filter-name>Servlet Event Bridge Filter</filter-name>
  <filter-class>org.jboss.seam.servlet.event.ServletEventBridgeFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>Servlet Event Bridge Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
  <filter-name>Catch Exception Filter</filter-name>
```

```
  <filter-class>org.jboss.seam.servlet.filter.CatchExceptionFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>Catch Exception Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

You're now ready to dive into the Servlet enhancements provided for you by the Seam Servlet module!

# Servlet event propagation

By including the Seam 3 Servlet module in your web application (and performing the necessary listener configuration for pre-Servlet 3.0 environments) you will also have the servlet lifecycle events propagated to the CDI event bridge so you can observe them in your beans. The event bridge works by installing a servlet listener and firing events on the BeanManager with associated qualifiers, passing along the event object.

## 2.1. Servlet context lifecycle listener

These events correspond to the `javax.servlet.ServletContextListener` interface. The event object fired is a `javax.servlet.ServletContext` (since it's the only relevant information in the `javax.servlet.ServletContextEvent` object). There are two qualifiers available that can be used for selecting the initialization or destruction of the servlet context.

| Qualifier | Description |
|---|---|
| @Initialized | Qualifies the creation event |
| @Destroyed | Qualifies the destruction event |

If you want to listen to both lifecycle events, leave out the qualifiers:

```
public void observeServletContext(@Observes ServletContext ctx)
{
  // Do something with the "servlet context" object
}
```

If you are interested in only a particular one, use a qualifer:

```
public void observeServletContextInitialized(@Observes @Initialized ServletContext ctx)
{
  // Do something with the "servlet context" object upon initialization
}
```

The name of the observer method is insignificant.

## 2.2. Application initialization

The `ServletContext` initialized event provides an excellent opportunity to perform startup logic as an alterative to using an EJB 3.1 startup singleton. Even better, you can configure the bean to be destroyed immediately following the initialization routine by leaving it as dependent scoped (dependent-scoped observers only live for the duration of the observe method invocation).

Here's an example of entering seed data into the database in a development environment (as indicated by a stereotype annotation named `@Development`).

```
@Stateless
@Development
public class SeedDataImporter
{
  @PersistenceContext
  private EntityManager em;

  public void loadData(@Observes @Initialized ServletContext ctx)
  {
    em.persist(new Product(1, "Black Hole", 100.0));
  }
}
```

If you'd rather not tie yourself to the Servlet API, you can observe the `WebApplication` type instead, which is an informational object provided by Seam Servlet that holds select information about the `ServletContext` such as the application name, context path, server info and start time.

```
public void loadData(@Observes @Initialized WebApplication webapp)
{
  ...
}
```

## 2.3. Servlet request lifecycle listener

These events correspond to the `javax.servlet.ServletRequestListener` inteface. The event object fired is a `javax.servlet.ServletRequest` (since it's the only relevant information in the `javax.servlet.ServletRequestEvent` object. There are two qualifiers available that can be used for selecting the initialization or destruction of the request.

| Qualifier | Description |
|---|---|
| @Initialized | Qualifies the initialization event |
| @Destroyed | Qualifies the destruction event |

If you want to listen to both lifecycle events, leave out the qualifiers.

```
public void observeRequest(@Observes ServletRequest request)
{
  // Do something with the servlet "request" object
```

```
}
```

If you are interested in only a particular one, use a qualifer

```
public void observeRequestInitialized(@Observes @Initialized ServletRequest request)
{
  // Do something with the servlet "request" object upon initialization
}
```

You can also listen specifically for a `javax.servlet.http.HttpServletRequest` simply by changing the expected event type.

```
public void observeRequestInitialized(@Observes @Initialized HttpServletRequest request)
{
  // Do something with the HTTP servlet "request" object upon initialization
}
```

The name of the observer method is insignificant.

## 2.4. Servlet response lifecycle listener

The Servlet API does not provide a listener for accessing the lifecycle of a response. Therefore, Seam Servlet simulates a response lifecycle listener using CDI events. These events parallel those provided by the `javax.servlet.ServletRequestListener` inteface. The event object fired is a `javax.servlet.ServletResponse`. There are two qualifiers available that can be used for selecting the initialization or destruction of the response.

| Qualifier | Description |
|---|---|
| @Initialized | Qualifies the initialization event |
| @Destroyed | Qualifies the destruction event |

If you want to listen to both lifecycle events, leave out the qualifiers.

```
public void observeResponse(@Observes ServletResponse response)
{
  // Do something with the servlet "response" object
}
```

If you are interested in only a particular one, use a qualifer

```
public void observeResponseInitialized(@Observes @Initialized ServletResponse response)
{
  // Do something with the servlet "response" object upon initialization
}
```

You can also listen specifically for a `javax.servlet.http.HttpServletResponse` simply by changing the expected event type.

```
public void observeResponseInitialized(@Observes @Initialized HttpServletResponse response)
{
  // Do something with the HTTP servlet "response" object upon initialization
}
```

The name of the observer method is insignificant.

## 2.5. Session lifecycle listener

These events correspond to the `javax.servlet.HttpSessionListener` interface. The event object fired is a `javax.servlet.http.HttpSession` (since it's the only relevant information in the `javax.servlet.http.HttpSessionEvent` object). There are two qualifiers available that can be used for selecting the initialization or destruction of the session.

| Qualifier | Description |
|---|---|
| @Initialized | Qualifies the creation event |
| @Destroyed | Qualifies the destruction event |

If you want to listen to both lifecycle events, leave out the qualifiers. Note that omitting all qualifiers will observe all events with a `HttpSession` as event object.

```
public void observeSession(@Observes HttpSession session)
{
  // Do something with the "session" object
}
```

If you are interested in only a particular one, use a qualifer

```
public void observeSessionInitialized(@Observes @Initialized HttpSession session)
{
  // Do something with the "session" object upon being initialized
```

```
}
```

The name of the observer method is insignificant.

## 2.6. Session activation listener

These events correspond to the `javax.servlet.HttpSessionActivationListener` interface. The event object fired is a `javax.servlet.http.HttpSession` (since it's the only relevant information in the `javax.servlet.http.HttpSessionEvent` object). There are two qualifiers available that can be used for selecting the activation or passivation of the session.

| Qualifier | Description |
|---|---|
| @DidActivate | Qualifies the activation event |
| @WillPassivate | Qualifies the passivation event |

If you want to listen to both lifecycle events, leave out the qualifiers. Note that omitting all qualifiers will observe all events with a `HttpSession` as event object.

```
public void observeSession(@Observes HttpSession session)
{
  // Do something with the "session" object
}
```

If you are interested in only a particular one, use a qualifer

```
public void observeSessionCreated(@Observes @WillPassivate HttpSession session)
{
  // Do something with the "session" object when it's being passivated
}
```

The name of the observer method is insignificant.

# Injectable Servlet objects and request state

Seam Servlet provides producers that expose a wide-range of information available in a Servlet environment (e.g., implicit objects such as ServletContext and HttpSession and state such as HTTP request parameters) as beans. You access this information by injecting the beans produced. This chapter documents the Servlet objects and request state that Seam Servlet exposes and how to inject them.

## 3.1. @Inject @RequestParam

The `@RequestParam` qualifier allows you to inject an HTTP request parameter (i.e., URI query string or URL form encoded parameter).

Assume a request URL of /book.jsp?id=1.

```
@Inject @RequestParam("id")
private String bookId;
```

The value of the specified request parameter is retrieved using the method `HttpServletRequest.getParameter(String)`. It is then produced as a dependent-scoped bean of type String qualified `@RequestParam`.

The name of the request parameter to lookup is either the value of the `@RequestParam` annotation or, if the annotation value is empty, the name of the injection point (e.g., the field name).

Here's the example from above modified so that the request parameter name is implied from the field name:

```
@Inject @RequestParam
private String id;
```

If the request parameter is not present, and the injection point is annotated with `@DefaultValue`, the value of the `@DefaultValue` annotation is returned instead.

Here's an example that provides a fall-back value:

```
@Inject @RequestParam @DefaultValue("25")
private String pageSize;
```

If the request parameter is not present, and the `@DefaultValue` annotation is not present, a null value is injected.

**Warning**

Since the bean produced is dependent-scoped, use of the `@RequestParam` annotation on class fields and bean properties is only safe for request-scoped beans. Beans with wider scopes should wrap this bean in an `Instance` bean and retrieve the value within context of the thread in which it's needed.

```
@Inject @RequestParam
private Instance<String> bookIdResolver;
...
String bookId = bookIdResolver.get();
```

## 3.2. @Inject ServletContext

The `ServletContext` is made available as an application-scoped bean. It can be injected safetly into any CDI bean as follows:

```
@Inject
private ServletContext context;
```

The producer obtains a reference to the `ServletContext` by observing the `@Initialized ServletContext` event raised by this module's Servlet-to-CDI event bridge.

## 3.3. @Inject HttpSession

The `HttpSession` is made available as a request-scoped bean. It can be injected safetly into any CDI bean as follows:

```
@Inject
private HttpSession session;
```

The producer obtains a reference to the `HttpSession` by observing the `@Initialized HttpServletRequest` event raised by this module's Servlet-to-CDI event bridge.

## 3.4. @Inject HttpServletRequest

The `HttpServletRequest` is made available as a request-scoped bean. It can be injected safetly into any CDI bean as follows:

```
@Inject
private HttpServletRequest request;
```

The producer obtains a reference to the `HttpServletRequest` by observing the `@Initialized HttpServletRequest` event raised by this module's Servlet-to-CDI event bridge.

## 3.5. @Inject @ContextPath

The context path is made available as a dependent-scoped bean. It can be injected safetly into any request-scoped CDI bean as follows:

```
@Inject @ContextPath
private String contextPath;
```

You can safetly inject the context path into a bean with a wider scope using an instance provider:

```
@Inject @ContextPath
private Instance<String> contextPathProvider;
...
String contextPath = contextPathProvider.get();
```

The context path is retrieved from the `HttpServletRequest`.

## 3.6. @Inject List<Cookie>

The list of `Cookie` objects is made available as a request-scoped bean. It can be injected safetly into any CDI bean as follows:

```
@Inject
private List<Cookie> cookies;
```

The producer uses a reference to the request-scoped `HttpServletRequest` bean to retrieve the `Cookie` intances by calling `getCookie()`.

# Servlet Context attribute BeanManager provider

Although discouraged as a general practice, there are circumstances access to the `BeanManager` is required outside of the CDI context. Seam Servlet includes a provider that retrieves the `BeanManager` from the Servlet context attribute named `javax.enterprise.inject.spi.BeanManager`, an alternative to the standard JNDI lookup mechanism defined in the JSR-299 specification. The Servlet module also handles binding the `BeanManager` to this attribute when the application is initialized. The work is performed in a CDI observer that is notified by the Servlet-CDI bridge provided by this very module.

Refer to the *BeanManager provider* chapter of the Weld Extensions reference guide for information on how to leverage the Servlet context provider to access the `BeanManager` from outside the CDI environment.