CHAPTER 3

Component Fundamentals

In this chapter, we're going to explore the core capabilities of Seam's component model, which is at the heart of all of the services provided by the framework. We'll do this in a practical way (of course), by building out a (slightly) more complete version of the Gadget Catalog application that I introduced in Chapter 1. As you move along in the chapter, you'll see me demonstrate the key capabilities of the Seam component model by adding these extensions to the Gadget Catalog application.

Caution Java EE combined with Seam is different from just plain Java EE. As you move along through this chapter and the ones that follow, I'll try to highlight the important differences between life in standard Java EE versus life in Java EE plus Seam. There will be times when this may seem a bit daunting, especially if you're not completely versant in Java EE. Stick with it and walk through the sample code I provide in the book's code bundle as you read this chapter. You won't be disappointed in the richer development framework that Seam provides once you master its various services and capabilities. It's a satisfying experience to master any framework that helps you do good things faster.

Seam Component Types

Seam provides a broader component model for Java EE, one that subsumes, in a sense, both the JSF and EJB component models. Note that Seam doesn't replace these other component models. Instead, it tries to unify them into a single component model that can be used (almost) interchangeably between JSF and EJB contexts.

If you put the various types of EJB components together with Plain Old Java Objects (POJOs) and JavaBeans supported by JSF, you have five options for implementing Seam components:

- JavaBeans/POJOs
- Stateless session EJBs

- Stateful session EJBs
- Entity EJBs
- Message-driven EJBs

It's important to understand from the outset that, while Seam makes it possible to use all of these component types in general, there are specific cases where the different types can be used and cases where they can't and/or shouldn't be used. Rather than describe each of these situations in an analytical fashion, I'll take a more practical approach (naturally). Let's look at some common development contexts in web applications and the types of Seam components that can be applied to them. The first two, form backing beans and action listeners, are directly supported by JSF, but native JSF only supports the use of regular JavaBeans for these, and the use of EJB components in these contexts is more complicated than you might like. The last context, browser-accessible components, is not supported directly by JSF at all. So here again, we're seeing how Seam both simplifies and extends Java EE.

Form Backing Beans

Backing beans are a common pattern used in Java web frameworks, including JSF and Struts. In JSF, backing beans are the targets of JSF forms and are also used to inject data into JSF pages. Standard JSF supports the use of JavaBeans/POJOs as form backing beans. The Seam component model extends this to allow EJB 3.0 components to serve as backing beans within JSF web applications as well.

If you think about the EJB component model for a minute, you'll realize that the best fit for backing beans are stateful session EJBs or entity EJBs. These types of EJB 3.0 components are meant to represent client state data, which aligns very well with their use as backing beans. If you really don't need the container services offered by EJB components, however, such as life-cycle management and instance pooling, plus persistence management in the case of entity EJBs, you'll probably want to stick to regular JavaBeans or POJOs for your backing beans.

You shouldn't use stateless session EJBs as backing beans, because their component models just don't make sense for this purpose. This isn't a limitation imposed by Seam or JSF, it's simply the nature of how stateless session EJBs are managed by their container. Stateless session beans are considered "shared property" among the clients hitting the application, with no association to any particular client. Backing beans, however, need to have an association with a single client session in order to be consistent with the transactions that the client is carrying out.

Message-driven EJBs are ruled out as backing beans for the same reason (the EJB container manages message-driven beans much like stateless session beans), but also because their purpose is to handle incoming JMS messages from clients. This isn't really