

J2EE and XDoclet

by

Table of contents

1 Defining EJBs.....	2
1.1 XDoclet Tags with Attributes.....	3
1.2 Sessions Beans.....	4
1.3 Entity Beans.....	4

1. Defining EJBs

The Java 2 Platform, Enterprise Edition (J2EE) platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs.

Any developer who wishes to write J2EE applications, will need to implement the application logic using a combination of some or all of the three types of Enterprise Java Beans defined in the J2EE model. Once the beans are implemented with interfaces, classes and deployment descriptors, they must then be deployed to an EJB container. An application server like JBoss AS is a good example of an EJB container.

- **Session Beans**

Performs a task for a client.

- **Entity Beans**

Represents a business entity object that exists in persistent storage.

- **Message-Driven Beans**

Acts as a listener for the Java Messaging Service API, processing messages asynchronously.

To implement entity and session enterprise beans, a developer needs to define the component interfaces, a bean class, and a primary key:

- *Remote interface*

The remote interface defines the bean's business methods that can be accessed from applications outside the EJB container: the business methods a bean presents to the outside world to do its work. It is used by the session and entity beans in conjunction with the remote home interface.

- *Remote home interface*

The home interface defines the bean's life-cycle methods that can be accessed from applications outside the EJB container: the life-cycle methods for creating new beans, removing beans, and finding beans. It is used by session and entity beans in conjunction with the remote interface.

- *Local interface*

Like the remote interface, but can be used by other beans co-located in the same EJB container. It allows beans to interact without the overhead of a distributed object protocol, which improves their performance.

- *Local home interface*

Like the remote home interface, but can be used by other beans co-located in the same EJB container. It allows beans to interact without the overhead of a distributed object protocol, which improves their performance.

- *Bean class*

The session and entity bean classes actually implement the bean's business and life-cycle methods.

- *Primary key*

The primary key is a very simple class that provides a pointer into the database. Only entity beans need a primary key.

Much of the information about how beans are managed at runtime is not addressed by the interfaces and classes described above e.g. security, transactions, naming, and other services as they are automatically handled by the EJB container. Still, the EJB container needs to know how to apply the primary services to each bean class at runtime, and to do this, *deployment descriptors* are used. These are simply XML documents.

In summary, to implement the simplest of beans, requires a developer to write up to 4 interfaces, 2 classes and a deployment descriptor. A developer can become very frustrated with this burden as it may seem like a duplication of effort to define business methods in the class, only to have to add them to two interfaces. Upon any change, there may be a requirement to change all of these files each time!

XDoclet saves developer's coding and debugging time through the use of tags with attributes inside javadoc documentation comments in the bean class's file. Tags are placed either at class level i.e. before the class definition or at method level i.e. before the method definition. XDoclet, initiated as a task with sub-tasks in an ant build, processes the tagged bean class and generates required interface, class and descriptor files.

1.1. XDoclet Tags with Attributes

```
/**
 * @ejb.bean
 *     name="BlogFacade"
 */
public abstract class BlogFacadeBean implements SessionBean {
    /**
     * @ejb.interface-method
     */
    public abstract String getId();
}
```

Class-level tag.

Class-level tag attribute.

Method-level tag with no attributes.

1.2. Sessions Beans

Session beans have much fewer attributes than entity beans. Below is an example class definition.

1.2.1. Stateless Session Bean Definition

```
/**
 * A session facade for managing a blog
 *
 * @ejb.bean
 *   name="BlogFacade"
 *   type="Stateless"
 *   view-type="remote"
 *   transaction-type="Container"
 *
 * @ejb.ejb-ref
 *   ejb-name="Blog"
 *   view-type="local"
 *   ref-name="ejb/BlogLocal"
 *
 * @ejb.ejb-ref
 *   ejb-name="Entry"
 *   view-type="local"
 *   ref-name="ejb/EntryLocal"
 *
 * @ejb.ejb-ref
 *   ejb-name="Topic"
 *   view-type="local"
 *   ref-name="ejb/TopicLocal"
 *
 * @ejb.security-role-ref
 *   role-name="Owner"
 *   role-link="blogowner"
 */
public abstract class BlogFacadeBean implements SessionBean {
    ...
}
```

1.3. Entity Beans

Entity beans are harder to define. Here is an example:

1.3.1. Entity Bean Definition

```
/**
 * An entity bean representing a blog (weblog).
 *
 * @ejb.bean
 *   name="Blog"
 *   type="CMP"
 *   cmp-version="2.x"
 *   primkey-field="id"
 *   view-type="local"
 *
 */
```

```
* @ejb.value-object
*
* @ejb.value-object
*   name="BlogSimple"
*   match="simple"
*
* @ejb.finder
*   signature="java.util.Collection findAll()"
*   query="SELECT OBJECT(b) FROM Blog AS b"
*/
public abstract class BlogBean implements EntityBean {
    ...
}
```