openejb-jar.xml

{scrollbar}

Overview

Geronimo uses the OpenEJB container for providing EJB services. With the advent of Java EE, the EJB container services such as transaction management, security, life cycle management can be declared in the EJB class itself using annotations. However, the EJB deployment descriptor can still be provided via usage of the **ejb-jar.xml** file. When both annotations and the **ejb-jar.xml** file are provided, the **ejb-jar.xml** file takes precedence over the annotations.

The Geronimo-specific deployment plan for an EJB application, which is usually packaged as an EJB JAR file, is called "**openejb-jar.xml**". The **openejb-jar.xml** deployment plan is used (in conjunction with the **ejb-jar.xml** Java EE deployment plan) to deploy enterprise applications to the Geronimo application server. The **openejb-jar.xml** deployment plan is an optional file, but is typically used when deploying a EJB JAR file. It is used to map roles and resources (e.g., security roles, EJB names, database resources, JMS resources, etc.) declared in the **openejb-jar.xml** deployment plan to corresponding entities deployed in the server. Also, if there are any EJB container-specific configurations required those setting are configured in this deployment plan as well. If the EJB module depends on any third party libraries or other services running in the server, all these third party libraries and the services are specified in the **openejb-jar.xml** file. Some EJB applications require class loading requirements different from the default class loading behavior. The **openejb-jar.xml** file allows the application deployer to configure this as well. There are many more configurations that could be done through the **openejb-jar.xml** file depending on the requirements of the EJB application.

Packaging

The **openejb-jar.xml** Geronimo-specific deployment plan can be packaged as follows:

1. Embedded in an EJB JAR file. In this case, the **openejb-jar.xml** file must be placed in the **/META-INF** directory of the JAR, which is the same place where the **ejb-jar.xml** file must be located.
2. Maintained separately from the EJB JAR file. In this case, the path to the file must be provided to the appropriate Geronimo deployer (e.g., command-line or console) when the EJB JAR file is deployed. Note that in this case, the filename may be named something other than **openejb-jar.xml** but must adhere to the same schema. Also note that this will not work if the EJB JAR file is to be embedded in an enterprise application EAR file (see below).
3. Embedded in an enterprise application EAR file: In one case, the high-level element **<ejb-jar>** can be embedded outside the EJB JAR file in the EAR file's **geronimo-application.xml** file
4. Embedded in an enterprise application EAR file: In another case, the actual **openejb-jar.xml** file can be embedded outside the EJB JAR file and placed elsewhere in the EAR and referenced with an **alt-dd** element in the **geronimo-application.xml** deployment plan file.

Schema

The **openejb-jar.xml** deployment plan is defined by the **openejb-jar-2.2.xsd** schema located in the **<geronimo_home>/schema/** subdirectory of the main Geronimo installation directory. The **openejb-jar-2.2.xsd** schema is shown here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2

Schema top-level elements

The root XML element in the **openejb-jar-2.2.xsd** schema is the **<openejb-jar-2.2>** element. The top-level XML elements of the **<openejb-jar>** root element are described in the sections below. The deployment plan should always use the OpenEJB namespace, and it typically requires elements from the Geronimo Naming, Geronimo Security, and Geronimo System namespaces. Additionally, it has a required attribute to identify its configuration name. A typical deployment for **openejb-jar.xml** can be presented as follows:

xmlopenejb-jar.xml Examplesolid <openejb-jar xmlns:openejb="http://openejb.apache.org/xml/ns/openejb-jar-2.2" targetNamespace="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:security="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:pkgen="http://openejb.apache.org/xml/ns/pkgen-2.1" version="1.0"> ... </openejb-jar>

<sys:environment>

The **<sys:environment>** XML element uses the Geronimo System namespace, which is used to specify the common elements for common libraries and module-scoped services, which is documented here:

- http://geronimo.apache.org/schemas-2.1/docs/geronimo-module-1.2.xsd.html

The **<sys:environment>** element contains the following elements:

- The **<moduleId>** element is used to provide the configuration name for the EJB application as deployed in the Geronimo server. It contains elements for the **groupId**, **artifactId**, **version** and module **type**. Module IDs are normally printed with slashes between the four components, such as **GroupID/ArtifactID/Version/Type**.

- The **<dependencies>** element is used to provide the configurations and third party libraries on which the EJB module is dependent upon. These configurations and libraries are made available to the EJB module via the Geronimo classloader hierarchy.

- The **<hidden-classes>** element can be used to provide some degree of control of the Geronimo classloader hierarchy, and mitigate clashes between classes loaded by the server and classes loaded by the EJB module. It is used to lists packages or classes that may be in a parent classloader, but must not be exposed to the EJB application. Since Geronimo is entirely open-source and utilizes many other open-source libraries it is possible that the server itself and the EJB application may have different requirements and/or priorities for the same open source project libraries. The **<hidden-classes>** element is typically used when the EJB application has requirements for a specific version of a library that is different than the version used by Geronimo itself. A simple example of this is when an EJB module uses, and most importantly includes, a version of the **Log4J** common logging library that is different than the version used by the Geronimo server itself. This might not provide the desired results. Thus, the **<hidden-classes>** element can be used to "hide" the Log4J classes loaded by all the parent classloaders of the EJB module, including those loaded by and for the Geronimo server itself, and only the Log4J classes included with the EJB module library will get loaded.

- The **<non-overridable-classes>** element can also be used to provide some degree of control of the Geronimo classloader hierarchy, but in the exact opposite manner than provided by the **<hidden-classes>** element. This element can be used to specify a list of classes or packages which will **only** be loaded from the parent classloader of the EJB module to ensure that the Geronimo server's version of a library is used instead of the version included with the EJB application.

- The **<inverse-classloading>** element can be used to specify that standard classloader delegation is to be reversed for this module. The Geronimo classloader delegation follows the Java EE 5 specifications, and the normal behavior is to load classes from a parent classloader (if available) before checking the current classloader. When the **<inverse-classloading>** element is used, this behavior is reversed and the current classloader will always be checked before looking in the parent classloader(s). This element is similar to the **<hidden-classes>** element since the desired behavior is to give the libraries packaged with the EJB JAR application (i.e., in META-INF/lib) precedence over anything used by the Geronimo server itself.

- The **<suppress-default-environment>** element can be used to suppress inheritance of environment by module (i.e., any default environment built by a Geronimo builder when deploying the plan will be suppressed). If the **<suppress-default-environment>** element is specified then any default environment build by a builder when deploying the plan will be suppressed. An example of where this is useful is when deploying a connector on an app client in a separate (standalone) module (not as part of a client plan). The connector builder defaultEnvironment includes some server modules that won't work on an app client, so you need to suppress the default environment and supply a complete environment including all parents for a non-app-client module you want to run on an app client. This element should not be used for EJB applications however.

An example **openejb-jar.xml** file is shown below using the **<sys:environment>** element:

xmlsolid<sys:environment> Example <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>openejb-jar-1</sys:artifactId> <sys:version>2.0</sys:version> <sys:type>ear</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.testsuite</sys:groupId> <sys:artifactId>agent-ds</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>tomcat6</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> </openejb-jar>

<sec:security>

The **<sec:security>** XML element uses the Geronimo Security namespace, and is documented here:

- http://geronimo.apache.org/schemas-2.1/docs/geronimo-security-2.0.xsd.html.

The **<sec:security>** element groups the security role mapping settings for the EJB application. This is an optional element, but if it is present all the EJB modules must make the appropriate access checks as outlined in the JACC specification. This element includes the **<role-mapping>** section that references the role(s) defined in the **<security-role>** element in the **application.xml** file.

<sys:gbean>

The **<sys:gbean>** XML element uses the Geronimo System namespace described at http://geronimo.apache.org/schemas-2.1/docs/geronimo-module-1.2.xsd.html.

The **<sys:gbean>** element is used to define GBean(s) that are configured and deployed with the EJB. These additional Geronimo services will be deployed when the application is deployed (and stopped when the application is stopped). Normally, the implementation classes for these services are included at the server level and referenced using a dependency element.

CMP Entity Beans

Container Managed Persistence (CMP) is still fully supported in OpenEJB/Geronimo, although the Java Persistence API is generally considered a better approach in Java EE. Apache Geronimo uses OpenJPA for providing Java Persistence API to Java EE applications deployed in the server. More information and details about the JPA deployment descriptor can be found here: Java Persistence API deployment plans.

<naming:cmp-connection-factory>

The **<naming:cmp-connection-factory>** XML element uses the Geronimo Naming namespace, which is used to identify the common elements for resolving EJB references, resource references, and Web services references, which is documented here:

- http://geronimo.apache.org/schemas-2.1/docs/geronimo-naming-1.2.xsd.html

The **<naming:cmp-connection-factory>** element is used to specify a JDBC connection pool that should be used by Container Managed Persistence (CMP) entity beans to connect to a database. Since the **<naming:cmp-connection-factory>** element points to a database pool using the same syntax a resource reference uses, there are multiple methods available to refer to the connection pool. It can be specified by a simple name using the **<resource-link>** element, by pattern using the **<pattern>** element, or finally by URL using the **<url>** element. The resource-link handles most common resource situations where the JDBC pools are deployed as J2EE connectors in the same application, or deployed standalone in the same server. But pattern or URL can be use for any. An example **openejb-jar.xml** using all three techniques is shown:

xmlsolid<naming:cmp-connection-factory> Example <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>openejb-jar-1</sys:artifactId> <sys:version>2.0</sys:version> <sys:type>ear</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.testsuite</sys:groupId> <sys:artifactId>agent-ds</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>tomcat6</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <!-- Begin configuration for Container Managed Persistence (CMP) --> <naming:cmp-connection-factory> <naming:pattern> <naming:groupId>resourcelocator-pattern-groupid</naming:groupId> <naming:artifactId>resourcelocator-pattern-artifactid</naming:artifactId> <naming:version>resourcelocator-pattern-version</naming:version> <naming:module>resourcelocator-pattern-module</naming:module> <naming:name>resourcelocator-pattern-name</naming:name> </naming:pattern> <naming:resource-link>resourcelocator-resourcelink</naming:resource-link> <naming:url>resourcelocator-url</naming:url> </naming:cmp-connection-factory> <!-- End configuration for Container Managed Persistence (CMP) --> </openejb-jar>

<ejb-ql-compiler-factory>

The **<ejb-ql-compiler-factory>** XML element uses the OpenEJB default namespace for a openejb-jar.xml file, which is documented here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2.xsd.html

The **<ejb-ql-compiler-factory>** element is used to specify the name of a Java class that can compile EJB-QL (Query Language) queries into SQL statements for a particular database product. This must be the fully-qualified class name of a class that implements **org.tranql.sql.EJBQLCompilerFactory**. The default is for the Derby database, which ships with Geronimo, although it may work for other database products as well. An example **openejb-jar.xml** using the **<ejb-ql-compiler-factory>** XML element is shown:

xmlsolid<ejb-ql-compiler-factory> Example <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>openejb-jar-1</sys:artifactId> <sys:version>2.0</sys:version> <sys:type>ear</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.testsuite</sys:groupId> <sys:artifactId>agent-ds</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>tomcat6</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <!-- Begin configuration for Container Managed Persistence (CMP) --> <naming:cmp-connection-factory> <naming:pattern> <naming:groupId>resourcelocator-pattern-groupid</naming:groupId> <naming:artifactId>resourcelocator-pattern-artifactid</naming:artifactId> <naming:version>resourcelocator-pattern-version</naming:version> <naming:module>resourcelocator-pattern-module</naming:module> <naming:name>resourcelocator-pattern-name</naming:name> </naming:pattern> <naming:resource-link>resourcelocator-resourcelink</naming:resource-link> <naming:url>resourcelocator-url</naming:url> </naming:cmp-connection-factory> <ejb-ql-compiler-factory>ejbqlcompilerfactory</ejb-ql-compiler-factory> <!-- End configuration for Container Managed Persistence (CMP) --> </openejb-jar>

<db-syntax-factory>

The **<db-syntax-factory>** XML element uses the OpenEJB default namespace for a openejb-jar.xml file, which is documented here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2

The **<db-syntax-factory>** element is used to specify the name of a Java class that can customize CMP SQL statements for a particular database product. This must be the fully-qualified class name of a class that implements **org.tranql.sql.DBSyntaxFactory**. The default is for the Derby

database, which ships with Geronimo, although it may work for other database products as well. An example **openejb-jar.xml** using the **<db-syntax-factory>** XML element is shown:

xmlsolid<db-syntax-factory> Example <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>openejb-jar-1</sys:artifactId> <sys:version>2.0</sys:version> <sys:type>ear</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.testsuite</sys:groupId> <sys:artifactId>agent-ds</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>tomcat6</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <!-- Begin configuration for Container Managed Persistence (CMP) --> <naming:cmp-connection-factory> <naming:pattern> <naming:groupId>resourcelocator-pattern-groupid</naming:groupId> <naming:artifactId>resourcelocator-pattern-artifactid</naming:artifactId> <naming:version>resourcelocator-pattern-version</naming:version> <naming:module>resourcelocator-pattern-module</naming:module> <naming:name>resourcelocator-pattern-name</naming:name> </naming:pattern> <naming:resource-link>resourcelocator-resourcelink</naming:resource-link> <naming:url>resourcelocator-url</naming:url> </naming:cmp-connection-factory> <ejb-ql-compiler-factory>ejbQlCompilerFactory</ejb-ql-compiler-factory> <db-syntax-factory>dbSyntaxFactory</db-syntax-factory> <!-- End configuration for Container Managed Persistence (CMP) --> </openejb-jar>

<enforce-foreign-key-constraints>

The **<enforce-foreign-key-constraints>** XML element uses the OpenEJB default namespace for a openejb-jar.xml file, which is documented here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2

The **<enforce-foreign-key-constraints>** element is effectively a true/false element – if it's present that means true, and if it's not present, that means false. If true, then Geronimo will make a special effort to execute insert, update, and delete statements in an order consistent with the foreign keys between tables. If false, then Geronimo will execute statements in any order, though still within the same transaction. This element should be present if the underlying database enforces foreign keys at the moment a statement is executed instead of at the end of the transaction. An example **openejb-jar.xml** setting the **<db-syntax-factory>** XML element to "true" is shown:

xmlsolid<enforce-foreign-key-constraints> Example <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>openejb-jar-1</sys:artifactId> <sys:version>2.0</sys:version> <sys:type>ear</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>org.apache.geronimo.testsuite</sys:groupId> <sys:artifactId>agent-ds</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> <sys:dependency> <sys:groupId>org.apache.geronimo.configs</sys:groupId> <sys:artifactId>tomcat6</sys:artifactId> <sys:version>2.1.1</sys:version> <sys:type>car</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <!-- Begin configuration for Container Managed Persistence (CMP) --> <naming:cmp-connection-factory> <naming:pattern> <naming:groupId>resourcelocator-pattern-groupid</naming:groupId> <naming:artifactId>resourcelocator-pattern-artifactid</naming:artifactId> <naming:version>resourcelocator-pattern-version</naming:version> <naming:module>resourcelocator-pattern-module</naming:module> <naming:name>resourcelocator-pattern-name</naming:name> </naming:pattern> <naming:resource-link>resourcelocator-resourcelink</naming:resource-link> <naming:url>resourcelocator-url</naming:url> </naming:cmp-connection-factory> <ejb-ql-compiler-factory>ejbQlCompilerFactory</ejb-ql-compiler-factory> <db-syntax-factory>dbSyntaxFactory</db-syntax-factory> <enforce-foreign-key-constraints/> <!-- End configuration for Container Managed Persistence (CMP) --> </openejb-jar>

<relationships>

The **<relationships>** XML element uses the OpenEJB default namespace for a openejb-jar.xml file, which is described here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2.

Container-managed relationships are initially defined in the **ejb-jar.xml** deployment descriptor, but the mappings to specific database elements are defined in the **openejb-jar.xml** file using the **<relationship>** element.

Common Elements for EJB Entity, Session, and Message-driven Beans

Unless otherwise noted, all the Common, EJB Entity, EJB Session, and EJB Message-driven elements use the OpenEJB default namespace for a openejb-jar.xml file, which is documented here:

- http://openejb.apache.org/xml/ns/openejb-jar-2.2

<enterprise-beans>

The **<enterprise-beans>** element used to specify references by **<entity>**, **<session>**, or **<message-driven>** EJB's. For example, a EJB Entity Bean would be specified similarly as below in an **openejb-jar.xml** file:

xmlsolid<entity> Example <enterprise-beans> <entity> <ejb-name>ExchangeRate</ejb-name> <local-jndi-name>ExchangeRate</local-jndi-name> <resource-ref> <ref-name>jdbc/BankDataSource</ref-name> <resource-link>BankPool</resource-link> </resource-ref> </entity> </enterprise-beans>

A EJB Session Bean would be specified similarly as below in an **openejb-jar.xml** file:

xmlsolid<session> Example <enterprise-beans> <session> <ejb-name>RetrieveEmployeeInfoBean</ejb-name> <business-remote>examples.session.stateless_dd.RetrieveEmployeeInfo</business-remote> <ejb-class>examples.session.stateless_dd.RetrieveEmployeeInfoBean</ejb-class> <session-type>Stateless</session-type> <transaction-type>Container</transaction-type> <resource-ref> <res-ref-name>jdbc/DataSource</res-ref-name> <res-type>javax.sql.DataSource</res-type> <res-auth>Container</res-auth> <res-sharing-scope>Shareable</res-sharing-scope> </resource-ref> </session> </enterprise-beans>

And an EJB Message-driven Bean would be specified similarly as below in an **openejb-jar.xml** file:

xmlsolid<message-driven> Example <enterprise-beans> <message-driven> <ejb-name>OrderRecvMDB</ejb-name> <resource-adapter> <resource-link>CommonConnectionFactory</resource-link> </resource-adapter> <activation-config> <activation-config-property> <activation-config-property-name>destination</activation-config-property-name> <activation-config-property-value>OrderQueue</activation-config-property-value> </activation-config-property> <activation-config-property> <activation-config-property-name>destinationType</activation-config-property-name> <activation-config-property-value>javax.jms.Queue</activation-config-property-value> </activation-config-property> </activation-config> </message-driven> </enterprise-beans>

<ejb-name>

The **<ejb-name>** element identifies the EJB that these settings apply to and must match the **<ejb-name>** for the EJB in **ejb-jar.xml** file.

<jndi-name>

The Home interface for the EJB is registered in JNDI at the address specified here. This global JNDI name is used by application clients to connect to this EJB. It is only meaningful if the EJB has a (remote) Home interface.

<local-jndi-name>

The LocalHome interface for the EJB is registered in JNDI at the address specified here. It is only meaningful if the EJB has a LocalHome interface.

<tssGroup>

The **<tssGroup>** set of elements is used to contains CORBA security settings (i.e., for EJBs exposed as CORBA objects). It is not necessary if the EJB will not be accessed via CORBA.

JNDI Environment References

All the JNDI reference elements in this section use the Geronimo Naming namespace, which is used to identify the common elements for resolving EJB references, resource references, and Web services references, and is documented here:

- http://geronimo.apache.org/schemas-2.1/docs/geronimo-naming-1.2.xsd.html

Additionally, more information and details about JNDI references can be found here: JNDI.

<naming:abstract-naming-entry>

The **<abstract-naming-entry>** element is not technically a JNDI reference element. It is included in this section because it is an abstract element used for containing these JNDI reference types:

- **<gbean-ref>**

  The **<gbean-ref>** element is used to map GBean references to GBeans configured outside the current module

- **<persistenceunit-ref>**

  The **<persistenceunit-ref>** element is used to map persistence unit references to persistence units configured outside the current module

- **<persistencecontext-ref>**

  The **<persistencecontext-ref>** element is used to map persistence context references to persistence contexts configured outside the current module

<naming:ejb-ref>

The **<naming:ejb-ref>** element is used to map EJB references to EJB's in other applications using remote home and remote interface. The application which contains the EJB being referenced should either be in same EAR or should be included in dependency list of this application. Also note as the EJB's referenced are in a different JVM all the client interfaces should also be included in the current application.

<naming:ejb-local-ref>

The **<naming:ejb-local-ref>** element is used to map EJB references to EJB's in other applications using local home and local interface. The application which contains the EJB being referenced should either be in same EAR or should be included in dependency list of this application. Also note as the EJB's referenced are in a different JVM all the client interfaces should also be included in the current application.

<naming:service-ref>

The **<naming:service-ref>** is used to map service references to service's in other applications. The application which contains the EJB being referenced should either be in same EAR or should be included in dependency list of this application.

<naming:resource-ref>

The **<naming:resource-ref>** element is used to map resource references to resources like JDBC resources, JMS resources, etc. configured outside the current application.

<naming:resource-env-ref>

The **<naming:resource-env-ref>** element is used to map resource references to administrative objects deployed as a part of connectors.

<naming:message-destination>

The **<naming:message-destination>** element is used to map resource references to a message-destination which is used within the deployed EJB application. These are typically a JMS queue or topic which acts like a destination for the messages delivered. Like all the JNDI references in this section, this element will not cause the creation of a message-destination, references an existing message-destination used within the deployed EJB application.

EJB Entity Beans

EJB Entity Beans settings can include JNDI names used by remote clients, CMP settings, and CORBA configurations and resolving references.

EJB Session Beans

An EJB session bean represents an interactive session between a single client and an application deployed on the server, and can also include JNDI names used by remote clients, CMP settings, and CORBA configurations and resolving references.

EJB Message-driven Beans

Message-driven bean settings include mapping the MDB to a specific message destination (usually a JMS Topic or Queue), as well as resolving references to other EJBs, resources, or Web services.

<naming:resource-adapter>

The **<naming:resource-adapter>** XML element uses the Geronimo Naming namespace, which is used to identify the common elements for resolving EJB references, resource references, and Web services references, and is documented here:

- http://geronimo.apache.org/schemas-2.1/docs/geronimo-naming-1.2.xsd.html

The **<naming:resource-adapter>** element identifies the resource adapter that this message-driven bean connects to. This is typically a JMS server, which is ActiveMQ for the default Geronimo JMS provider. It identifies the resource adapter instance that this MDB should use to connect to its destination. For example, a specific ActiveMQ broker may have several resource adapter instances set up, with different authentication settings, and this identifies the specific instance to use. Like the **<naming:cmp-connection-factory>**, there are multiple methods available to refer to the ActiveMQ broker. It can be specified by a simple name using the **<resource-link>** element, by pattern using the **<pattern>** element, or finally by URL using the **<url>** element. The resource-link handles most common resource situations (e.g., a JMS resource adapter deployed as part of the same EAR or in the top level of the server), but pattern or URL can be use for any. This might be important if, for example, two resource adapter deployments use the same name, so that the **<resource-link>** does not uniquely identify one and it must be fully-qualified. This can be used to identify any resource adapter in the same EAR or at the top level in the server. The value specified here should match the **<resourceadapter-name>** specified for the resource adapter instance in its Geronimo deployment plan.

<activation-config>

The **<activation-config>** XML element is used to specify any configuration data (in the form of name/value pairs) required by the resource adapter in order to supply messages to the MDB. For example:

xmlsolid<activation-config> Example <activation-config> <activation-config-property> <activation-config-property-name>destination</activation-config-property-name> <activation-config-property-value>OrderQueue</activation-config-property-value> </activation-config-property> <activation-config-property> <activation-config-property-name>destinationType</activation-config-property-name> <activation-config-property-value>javax.jms.Queue</activation-config-property-value> </activation-config-property> </activation-config>