Deployment plans

{scrollbar}

Every module that you install in Geronimo, whether it is a service, application, resource, etc., can be configured via a deployment plan. These deployment plans are XML files based on XML Schemas containing the configuration details for a specific application module or component. The Java EE 5 specification defines standard deployment descriptors such as `web.xml`, `application.xml` and so on. In some cases, the deployment descriptor is all that is required to install a module into a Geronimo server. However, in many cases, server-specific configuration is required when modules are installed. This server-specific configuration is accomplished by using Geronimo deployment plans.

Geronimo deployment plans can be packaged along with the application or specified externally at deployment time. If provided during deployment, this plan will overwrite any other Geronimo specific deployment plan provided with the application.

To package the deployment plans in you application you have to follow some naming conventions and place the file in a specific directory within your packaged application. For example, in a web application you would include the `geronimo-web.xml` under the `/WEB-INF` directory, the same place where you are also providing the `web.xml` descriptor, all within the WAR. For an enterprise application you would include the `geronimo-application.xml` under the `/META-INF` directory, the same place where you are also providing the `application.xml` descriptor, all within the WAR.

The Java EE 5 specification also let you use Annotations where you add resource references, dependencies, etc. directly in the code. Geronimo provides a Plan Creator that automatically generates the necessary deployment plans based on the standard deployment descriptors and annotations.

This document is organized in the following sections:

XML Schemas

Java EE Deployment Plans

Module Type	Geronimo Schema	Preferred Java EE Schema
General (Tomcat or Jetty) Web Application (WAR)	http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1	web-app_2_5.xsd
Tomcat-Only Web Application (WAR)	http://geronimo.apache.org/xml/ns/j2ee/web/tomcat-2.0.1	web-app_2_5.xsd
Jetty-Only Web Application (WAR)	http://geronimo.apache.org/xml/ns/j2ee/web/jetty-2.0.2	web-app_2_5.xsd
EJB (JAR)	http://openejb.apache.org/xml/ns/openejb-jar-2.2	ejb-jar_3_0.xsd
J2EE Connector (RAR)	http://geronimo.apache.org/xml/ns/j2ee/connector-1.2	connector_1_5.xsd
Application Client (JAR)	http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0	application-client_5.xsd
Application (EAR)	http://geronimo.apache.org/xml/ns/j2ee/application-2.0	application_5.xsd

Common Elements & Configuration

Module Type	Geronimo Schema	Description
Server Plans & Common Elements	http://geronimo.apache.org/xml/ns/deployment-1.2	Used to deploy new services in Geronimo in a standalone plan, and also contains common elements used by many other plans.
Geronimo Plugin Descriptor	http://geronimo.apache.org/xml/ns/plugins-1.3	Metadata on a Geronimo plugin or a list of available Geronimo plugins.
Security Mapping	http://geronimo.apache.org/xml/ns/security-2.0	Common security elements used by other plans.
Security Realms	http://geronimo.apache.org/xml/ns/loginconfig-2.0	Abbreviated syntax for configuring security realm and login module GBeans. You can either manually configure multiple GBeans or declare a single GBean for the realm using this to configure all the login modules.
Naming	http://geronimo.apache.org/xml/ns/naming-1.2	Common elements for references to other components (EJBs, database pools, JMS resources, J2EE Connectors, Web Services, etc.)
Primary Key Generator	http://openejb.apache.org/xml/ns/pkgen-2.1	Abbreviated syntax for configuring primary key generators for CMP entity beans. Avoids manually configuring and wiring up PK generator GBeans.
CORBA CSS Configuration	http://openejb.apache.org/xml/ns/corba-css-config-2.1	Abbreviated syntax for configuring security for clients accessing remote EJBs via CORBA.
CORBA TSS Configuration	http://openejb.apache.org/xml/ns/corba-tss-config-2.1	Abbreviated syntax for configuring security for EJBs exposed via CORBA.
config.xml	http://geronimo.apache.org/xml/ns/attributes-1.2	The format of the `var/config/config.xml` file.

Tomcat Web App Configuration	http://geronimo.apache.org/xml/ns/web/tomcat/config-1.0	If you use the generic (`geronimo-web-2.0.xsd`) web application configuration, you can use these elements in the `container-config` element to configure Tomcat-specific behavior.
Jetty Web App Configuration	http://geronimo.apache.org/xml/ns/web/jetty/config-1.0.1	If you use the generic (`geronimo-web-2.0.xsd`) web application configuration, you can use these elements in the `container-config` element to configure Jetty-specific behavior.

Configurations

In this section, we will discuss about the configurations that are already deployed and running in the server when the server is installed and started.

Connection pools

Apache Geronimo ships with embedded Derby database and ActiveMQ message broker. There are also connection pools that connect to Derby and activeMQ configured to run in the installed server. The following sections discuss about various such configurations already running in the installed server.

Embedded Derby Database connection pool

Apache Geronimo ships with embedded Derby database. The Derby libraries are present in the server repository at `<geronimo_home>/repository/org/apache/derby`. By default, a Derby database by name `SystemDatabase` is created and the files related to the database are stored at `<geronimo_home>/var/derby/SystemDatabase`. Along with that, by default, server deploys a database connection pool over the `SystemDatabase` with the configuration name `org.apache.geronimo.configs/system-database/2.1/car`. The name of the database connection pool is `SystemDatasource`. The configuration artifacts are stored at `<geronimo_home>/repository/org/apache/geronimo/configs/system-database`. The deployment plan used for database connection pool is as follows.

XMLsolidSystemDatasource <?xml version="1.0" encoding="UTF-8"?> <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>system-database</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>car</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>transaction</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>car</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.apache.geronimo.modules</dep:groupId> <dep:artifactId>geronimo-derby</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>jar</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.apache.geronimo.modules</dep:groupId> <dep:artifactId>geronimo-timer</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>jar</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.apache.derby</dep:groupId> <dep:artifactId>derby</dep:artifactId> <dep:version>10.2.2.0</dep:version> <dep:type>jar</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.apache.derby</dep:groupId> <dep:artifactId>derbynet</dep:artifactId> <dep:version>10.2.2.0</dep:version> <dep:type>jar</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.apache.derby</dep:groupId> <dep:artifactId>derbyclient</dep:artifactId> <dep:version>10.2.2.0</dep:version> <dep:type>jar</dep:type> </dep:dependency> <dep:dependency> <dep:groupId>org.tranql</dep:groupId> <dep:artifactId>tranql-connector-derby-embed-xa </dep:artifactId> <dep:version>1.3</dep:version> <dep:type>rar</dep:type> </dep:dependency> </dep:dependencies> <dep:hidden-classes/> <dep:non-overridable-classes/> </dep:environment> <resourceadapter> <outbound-resourceadapter> <connection-definition> <connectionfactory-interface>javax.sql.DataSource </connectionfactory-interface> <connectiondefinition-instance> <name>SystemDatasource</name> <config-property-setting name="UserName"/> <config-property-setting name="Password"/> <config-property-setting name="DatabaseName"> SystemDatabase </config-property-setting> <config-property-setting name="CreateDatabase"> true </config-property-setting> <connectionmanager> <xa-transaction> <transaction-caching/> </xa-transaction> <single-pool> <max-size>100</max-size> <blocking-timeout-milliseconds> 5000 </blocking-timeout-milliseconds> <select-one-assume-match/> </single-pool> </connectionmanager> </connectiondefinition-instance> <connectiondefinition-instance> <name>NoTxDatasource</name> <config-property-setting name="UserName"/> <config-property-setting name="Password"/> <config-property-setting name="DatabaseName"> SystemDatabase </config-property-setting> <config-property-setting name="CreateDatabase"> true </config-property-setting> <connectionmanager> <no-transaction/> <single-pool> <max-size>100</max-size> <blocking-timeout-milliseconds>5000 </blocking-timeout-milliseconds> <select-one-assume-match/> </single-pool> </connectionmanager> </connectiondefinition-instance> </connection-definition> </outbound-resourceadapter> </resourceadapter> <gbean name="DerbySystem" class="org.apache.geronimo.derby.DerbySystemGBean"> <reference name="ServerInfo"> <name>ServerInfo</name> </reference> <attribute name="derbySystemHome">var/derby</attribute> </gbean> <gbean name="DerbyNetwork" class="org.apache.geronimo.derby.DerbyNetworkGBean"> <reference name="derbySystem"> <name>DerbySystem</name> </reference> <attribute name="host">localhost</attribute> <attribute name="port">1527</attribute> </gbean> <gbean name="DerbyLog" class="org.apache.geronimo.derby.DerbyLogGBean"> <reference name="DerbySystem"> <name>DerbySystem</name> </reference> </gbean> <gbean name="DerbyDriver" class="org.apache.geronimo.system.util.JDBCDriverRegistrationGBean"> <attribute name="driverClassName"> org.apache.derby.jdbc.EmbeddedDriver </attribute> </gbean> <gbean name="DerbyClientDriver" class="org.apache.geronimo.system.util.JDBCDriverRegistrationGBean"> <attribute name="driverClassName"> org.apache.derby.jdbc.ClientDriver </attribute> </gbean> <gbean name="TransactionalThreadPooledTimer" class="org.apache.geronimo.timer.jdbc.JDBCStoreThreadPooledTransactionalTimer"> <attribute name="repeatCount">5</attribute> <reference name="TransactionManager"> <name>TransactionManager</name> </reference> <reference name="ManagedConnectionFactoryWrapper"> <name>SystemDatasource</name> </reference> <reference name="ThreadPool"> <name>DefaultThreadPool</name> </reference> <dependency> <name>DerbySystem</name> </dependency> </gbean> <gbean name="NonTransactionalThreadPooledTimer" class="org.apache.geronimo.timer.jdbc.JDBCStoreThreadPooledNonTransactionalTimer"> <reference name="TransactionManager"> <name>TransactionManager</name> </reference> <reference name="ManagedConnectionFactoryWrapper"> <name>SystemDatasource</name> </reference> <reference name="ThreadPool"> <name>DefaultThreadPool</name> </reference> <dependency> <name>DerbySystem</name> </dependency> </gbean> </connector>
The default namespace of the above XML document is `http://geronimo.apache.org/xml/ns/j2ee/connector-1.2`. The XML elements that do not have a namespace prefix belong to the default namespace.

After starting the server, the running database connection pool `SystemDatasource` can be observed on the admin console from `console Navigation => Services => Database pools`. The resource adapter used to deploy the above database connection pool is `tranql-connector-derby-embed-xa-1.3.rar`. The above plan is actually deployment plan of a outbound resource adapter. If the above plan is packaged along with the `ra` file, the xml content will be placed in `META-INF/geronimo-ra.xml` of the archive.

Closely observe various configurations in the deployment plan. Many derby libraries in the server repository are mentioned as dependencies. After configuring the outbound resource adapter, there are series of gbeans configured for the database connection pool.

Embedded ActiveMQ resource adapter

By default, a JMS resource adapter that connects to embedded activemq message broker is deployed and running in the apache geronimo server. This is an outbound jms resource adapter that configures a connection factory and two message queues. The configuration name of the resource adapter is `org.apache.geronimo.configs/activemq-ra/2.1/car`. The artifacts of the resource adapter are stored at `<geronimo_home>/repository/org/apache/geronimo/configs/activemq-ra`. The deployment plan is as follows.

XMLsolidActiveMQ RA <?xml version="1.0" encoding="UTF-8"?> <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-ra</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>car</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-broker</dep:artifactId> <dep:version>2.1</dep:version> <dep:type>car</dep:type> </dep:dependency> </dep:dependencies> <dep:hidden-classes/> <dep:non-overridable-classes/> </dep:environment> <resourceadapter> <resourceadapter-instance> <resourceadapter-name>ActiveMQ RA</resourceadapter-name> <config-property-setting name="ServerUrl">tcp://0.0.0.0:61616</config-property-setting> <config-property-setting name="UserName">geronimo</config-property-setting> <config-property-setting name="Password">geronimo</config-property-setting> <workmanager> <gbean-link>DefaultWorkManager</gbean-link> </workmanager> </resourceadapter-instance> <outbound-resourceadapter> <connection-definition> <connectionfactory-interface>javax.jms.ConnectionFactory</connectionfactory-interface> <connectiondefinition-instance> <name>DefaultActiveMQConnectionFactory</name> <implemented-interface>javax.jms.QueueConnectionFactory</implemented-interface> <implemented-interface>javax.jms.TopicConnectionFactory</implemented-interface> <connectionmanager> <xa-transaction> <transaction-caching/> </xa-transaction> <single-pool> <max-size>10</max-size> <blocking-timeout-milliseconds>5000</blocking-timeout-milliseconds> <match-one/> </single-pool> </connectionmanager> </connectiondefinition-instance> </connection-definition> </outbound-resourceadapter> </resourceadapter> <adminobject> <adminobject-interface>javax.jms.Queue</adminobject-interface> <adminobject-class>org.apache.activemq.command.ActiveMQQueue</adminobject-class> <adminobject-instance> <message-destination-name>MDBTransferBeanOutQueue</message-destination-name> <config-property-setting name="PhysicalName">MDBTransferBeanOutQueue</config-property-setting> </adminobject-instance> </adminobject> <adminobject> <adminobject-interface>javax.jms.Queue</adminobject-interface> <adminobject-class>org.apache.activemq.command.ActiveMQQueue</adminobject-class> <adminobject-instance> <message-destination-name>SendReceiveQueue</message-destination-name> <config-property-setting name="PhysicalName">SendReceiveQueue</config-property-setting> </adminobject-instance> </adminobject> </connector>
The default namespace of the deployment plan is `http://geronimo.apache.org/xml/ns/j2ee/connector-1.2`. The xml elements that do not have a namespace prefix belong to default namespace.

The resource adapter used to deploy the above plan is `<geronimo_home>/repository/org/apache/geronimo/modules/geronimo-activemq-ra/2.1`. After the server is started, the running resource adapter can be looked up on the admin console from `Console Navigation => Services => JMS Resource`. We can also observe the connection factories and queues deployed by the resource adapter on the admin console.

Security

A Java EE application may consist of several components that can be deployed into different containers such as WEB container, EJB container, WebServices container in a JEE5 server. This kind of deployment allows multi-tier applications that interact with one another to perform a given user task. Multi-tier JEE5 applications can be secured by properly selecting authenticating mechanisms and designing authorization levels or roles. If the application components use declarative security management, the authentication and authorization aspects are declared in corresponding JEE5 deployment descriptors. The declared security roles or levels are mapped to real security roles or levels in the geronimo deployment plans through security realms. In Apache Geronimo , the security realms abstract away authentication and authorization aspects of the application components. The authentication and authorization together enable access control for the various components of the application.

Depending on the selected authenticating system, a JAAS login module is selected and configured in a security realm. JAAS login modules connect to corresponding user/group repositories and perform authentication and retrieve authorization information. The Geronimo server provides login modules that connect to different types of user/group repositories. These are **PropertiesFileLoginModule**, **LDAPLoginModule**, **SQLLoginModule** and **CertificatePropertiesFileLoginModule**.

For example, Geronimo uses **geronimo-admin** security realm to authenticate users when they login to the geronimo administration Console. The deployment plan of the security realm is follows.

geronimo-admin security realm

xmlsolidgeronimo-admin security realm <module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2"> <environment> <moduleId> <groupId>console.realm</groupId> <artifactId>geronimo-admin</artifactId> <version>1.0</version> <type>car</type> </moduleId> <dependencies> <dependency> <groupId>org.apache.geronimo.framework</groupId> <artifactId>j2ee-security</artifactId> <type>car</type> </dependency> </dependencies> </environment> <gbean name="geronimo-admin" class="org.apache.geronimo.security.realm.GenericSecurityRealm" xsi:type="dep:gbeanType" xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <attribute name="realmName">geronimo-admin</attribute> <reference name="ServerInfo"> <name>ServerInfo</name> </reference> <xml-reference name="LoginModuleConfiguration"> <log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-2.0"> <log:login-module control-flag="REQUIRED" wrap-principals="false"> <log:login-domain-name>geronimo-admin</log:login-domain-name> <log:login-module-class> org.apache.geronimo.security.realm.providers.PropertiesFileLoginModule </log:login-module-class> <log:option name="groupsURI">var/security/groups.properties</log:option> <log:option name="usersURI">var/security/users.properties</log:option> </log:login-module> </log:login-config> </xml-reference> </gbean> </module>
The default namespace of the above XML document is `http://geronimo.apache.org/xml/ns/deployment-1.2`. The XML elements that do not have a namespace prefix belong to the default namespace.

The above security realm is deployed over two property files `<geronimo_home>/var/security/users.properties` and `var/security/groups.properties` that contain user/group information using `org.apache.geronimo.security.realm.providers.PropertiesFileLoginModule`. The Geronimo Administration Console is a web application that uses the above security realm for user authentication.

The security realm deployment plan is an XML file that uses http://geronimo.apache.org/xml/ns/deployment-1.2 schema for **moduleId**, dependency and security realm GBean configurations. The XML file uses http://geronimo.apache.org/xml/ns/loginconfig-2.0 schema for login module configuration. All the XML schema files (`.xsd`) are located at `<geronimo_home>/schema` directory.

The following table provides the summary of user/group repositories and corresponding login modules in Apache Geronimo

User/Group Repository	LoginModule
Property files	`org.apache.geronimo.security.realm.providers.PropertiesFileLoginModule`
Database	`org.apache.geronimo.security.realm.providers.SQLLoginModule`
Ldap repository	`org.apache.geronimo.security.realm.providers.LDAPLoginModule`
Certificate Repository	`org.apache.geronimo.security.realm.providers.CertificatePropertiesFileLoginModule`
Any other	`User has to supply the custom JAAS module. Admin console can be used to deploy a security realm over custom JAAS login modules`

Depending on the type of the login module, the options for configuration may change.

Once a security realm is deployed, it is available for any JEE5 application deployed in Geronimo to map declared roles to actual users/groups through a Geronimo specific deployment plan.

Applications

An enterprise application archive (EAR) can consist of several application modules. The application modules can be several web application archives (WAR) , EJB modules (JAR), application client modules (JAR) or resource archive modules (RAR). User can either deploy these modules individually or bundle them into a single EAR file and deploy that file.

When deployed individually, each application module should accompany a Geronimo deployment plan to map declared resources names, ejb names, security roles, JMS roles (if any) to actual resources in the server. The Geronimo deployment plans also contain any Geronimo specific settings and configurations. When deployed as a single bundle (EAR), user can create a single Geronimo deployment plan accomplish to perform all the mappings /settings and configurations.

The following table summarizes different JEE5 modules and corresponding Geronimo deployment plans accompany them.

JEE module	JEE deployment descriptor (DD)	Geronimo deployment plan
web application archive (WAR)	`web.xml`	geronimo-web.xml
EJB application archive (JAR)	`ejb-jar.xml`	openejb-jar.xml
resource adapter archive (RAR)	`ra.xml`	geronimo-ra.xml
enterprise application archive (EAR)	`application.xml`	geronimo-application.xml
enterprise application client archive (JAR)	`application-client.xml`	geronimo-application-client.xml

Web Application deployment plan

In the `geronimo-web.xml` file, application deployer maps the security roles, ejb names, database resources, JMS resources, etc. declared in `web.xml` to corresponding entities deployed in the server. In addition to that, if there are any web container specific configurations, such as Tomcat or Jetty specific, depending on the application needs, all these settings are configured as well here. If the web application depends on any third party libraries or other services running in the server, all these dependencies are declared in the plan. Some web applications require class loading requirements different from the default class loading behavior. The `geronimo-web.xml` allows application deployer to configure this as well. There are many more configurations that could be done through `geronimo-web.xml` depending on the needs of web application. The following sections briefly explain how `geronimo-web.xml` can be used to configure the web container and web applications.

The `geronimo-web.xml` uses XML elements from http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1 namespace and one or more namespaces mentioned in Common elements and Configuration section earlier in the document.

For example, the following `web.xml` and `geronimo-web.xml` are the deployment descriptor and Geronimo deployment plan respectively, of a web application that connects to a datasource deployed on DB2 and retrieves data from a table.

xmlsolidweb.xml <?xml version="1.0" encoding="ISO-8859-1"?> <web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5"> <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> <welcome-file-list> <welcome-file>jsp/EMPdemo.jsp</welcome-file> </welcome-file-list> </web-app>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace.

With Servlet 2.5 specification, many of the declarations done through `web.xml` can also be done through corresponding annotations in the servlets and JSPs. When both annotations and web.xml are provided, the declarations in web.xml takes precedence over annotations.

The web module connects to back end datasource using its JNDI name `jdbc/DataSource` as declared in the `web.xml`.

xmlsolidgeronimo-web.xml <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>samples</sys:groupId> <sys:artifactId>EmployeeDemo</sys:artifactId> <sys:version>2.5</sys:version> <sys:type>war</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>samples</sys:groupId> <sys:artifactId>EmployeeDatasource</sys:artifactId> <sys:version>2.5</sys:version> <sys:type>rar</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <context-root>/EmployeeDemo</context-root> <naming:resource-ref> <naming:ref-name>jdbc/DataSource</naming:ref-name> <naming:resource-link>jdbc/EmployeeDatasource</naming:resource-link> </naming:resource-ref> </web-app>
The default namespace of the above XML document is `http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1`. The XML elements that do not have a namespace prefix belong to the default namespace.

Observe the various XML tags and corresponding namespaces used in the deployment plan for various purposes.

`<sys:environment> .. </sys:environment>` : These elements provide the moduleid configuration and the dependencies. The moduleId elements provide the configuration name for the web module. So, when the web module is deployed, it is given the configuration name `samples/samples/2.5/jar`. The dependencies elements provide the configurations and third party libraries on which the web module is dependent on. These configurations and libraries will be available to the web module via a classloader hierarchy. In this case, the web module is dependent on `samples/EmployeeDatasource/2.5/rar` which is the configuration of the deployed Datasource that connects to a back end DB2 database. The Datasource deploys a database connection pool (`javax.sql.Datasource`) with name `jdbc/EmployeeDatasource`.

`<sys:context-root> .. </sys:context-root>` : The XML elements used to provide the web context root of the web applications.

`<naming:resource-ref> .. </naming:resource-ref>` : These elements help us to configure the resource references. In this case, the datasource reference `jdbc/DataSource` is mapped to `jdbc/EmployeeDatasource`.

In the `EMPdemo.jsp`, the following java code snippet is used to obtain a connection from the datasource.

JAVAsolidEMPdemo.jsp .... Context initContext = new InitialContext(); Context envContext = (Context)initContext.lookup("java:comp/env"); DataSource ds = (DataSource)envContext.lookup("jdbc/DataSource"); Connection con = ds.getConnection(); ....

The above descriptor and the plan files are the simple illustrations that explain how web modules are developed and assembled for Apache Geronimo. Similarly, many other configurations can be performed in the `geronimo-web.xml`.

All the XML schema files are located at `<geronimo_home>/schema` directory. Please go through the `.xsd` files to have a feel of XML tags that can be used in `geronimo-web.xml` for configuring web applications.

EJB Application deployment plan

Geronimo uses OpenEJB container for providing ejb services. With the advent of JEE 5, 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 through `ejb-jar.xml` file. When both annotations and `ejb-jar.xml` file are provided, the `ejb-jar.xml` file takes precedence over the annotations.

The `openejb-jar.xml` file contains deployment plan for ejb modules. In the `openejb-jar.xml` file, the application deployer maps the security roles, ejb names, database resources, JMS resources, etc. declared in `ejb-jar.xml` file to corresponding entities deployed in the server. In addition to that, if there are any ejb container specific configurations to be done, the required settings are configured as well here. 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 application deployer to configure this as well. There are many more configurations that could be done through `openejb-jar.xml` file depending on the needs of the ejb application. The following sections briefly explain how `openejb-jar.xml` file can be used to configure the ejb container and ejb applications.

For example, the below XML content is the deployment descriptor (`ejb-jar.xml`) of a stateless session bean that connects to a back end DB2 database.

XMLsolidejb-jar.xml <?xml version="1.0" encoding="UTF-8" ?> <ejb-jar xmlns="http://java.sun.com/xml/ns/javaee" version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"> <description>Stateless Session Bean Example</description> <display-name>Stateless Session Bean Example</display-name> <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> <interceptors> <interceptor> <interceptor-class> examples.session.stateless_dd.RetrieveEmployeeInfoCallbacks </interceptor-class> <post-construct> <lifecycle-callback-method>construct</lifecycle-callback-method> </post-construct> <post-activate> <lifecycle-callback-method>activate</lifecycle-callback-method> </post-activate> <pre-passivate> <lifecycle-callback-method>passivate</lifecycle-callback-method> </pre-passivate> </interceptor> </interceptors> <assembly-descriptor> <interceptor-binding> <ejb-name>RetrieveEmployeeInfoBean</ejb-name> <interceptor-class> examples.session.stateless_dd.RetrieveEmployeeInfoCallbacks </interceptor-class> </interceptor-binding> </assembly-descriptor> </ejb-jar>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace.

In EJB3.0, most of the deployment descriptor declarations can be done through the corresponding annotations in the bean class. However, if a deployment descriptor is supplied (`ejb-jar.xml`), the declarations in the deployment descriptor will override the annotations.

The ejb module connects to back end datasource using its JNDI name `jdbc/DataSource` as declared in the `ejb-jar.xml`. It also declares that the ejb is a stateless session bean and provides an interceptor class for the bean. The interceptor class will have callback methods which container calls when the corresponding events occur in the bean's life cycle.

For the above deployment descriptor, we will have to provide a corresponding deployment plan file (`openejb-jar.xml`) that maps the declared datasource to actual datasource deployed in the server. The following is the deployment plan.

XMLsolidopenejb-jar.xml <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>samples</sys:groupId> <sys:artifactId>EmployeeDemo-ejb-dd</sys:artifactId> <sys:version>3.0</sys:version> <sys:type>jar</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>console.dbpool</sys:groupId> <sys:artifactId>jdbc/FEmployeeDatasource</sys:artifactId> <sys:version>1.0</sys:version> <sys:type>rar</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <enterprise-beans> <session> <ejb-name>RetrieveEmployeeInfoBean</ejb-name> <naming:resource-ref> <naming:ref-name>jdbc/DataSource</naming:ref-name> <naming:resource-link>jdbc/EmployeeDatasource</naming:resource-link> </naming:resource-ref> </session> </enterprise-beans> </openejb-jar>
The default namespace of the above XML document is `http://openejb.apache.org/xml/ns/openejb-jar-2.2`. The XML elements that do not have a namespace prefix belong to the default namespace.

Observe the various XML tags and corresponding namespaces used in the deployment plan for various purposes.

`<sys:environment> .. </sys:environment>` : These elements provide the moduleId configuration and the dependencies. The moduleId elements provide the configuration name for the ejb module. So, when the ejb module is deployed, it is given the configuration name `samples/EmployeeDemo-ejb-dd/3.0/jar`. The dependencies elements provide the configurations and third party libraries on which the ejb module is dependent on. These configurations and libraries will be available to the ejb module via a classloader hierarchy. In this case, the ejb module is dependent on `console.dbpool/jdbc/FEmployeeDatasource/1.0/jar` which is the configuration of the deployed Datasource that connects to a back end DB2 database. The Datasource deploys a database connection pool (`javax.sql.Datasource`) with name `jdbc/EmployeeDatasource`.

`<enterprise-beans> .. </enterprise-beans>` : These elements help us to configure the enterprise beans. In this case, the datasource reference `jdbc/DataSource` is mapped to `jdbc/EmployeeDatasource`.

In the ejb bean class, the following java code is used to obtain a connection from the datasource.

JAVAsolidexamples.session.stateless_dd.RetrieveEmployeeInfoBean.java .... Context initContext = new InitialContext(); Context envContext = (Context) initContext.lookup("java:comp/env"); DataSource ds = (DataSource)envContext.lookup("jdbc/DataSource"); Connection con = ds.getConnection(); ....

The above descriptor and plan are the simple illustrations that explain how ejb modules are developed and assembled for Apache Geronimo. Similarly, many other configurations can be performed in the `openejb-jar.xml`. The schema for the plan is openejb-jar-2.1.xsd

Enterprise application deployment plan

An enterprise application archive (EAR) can consist of many sub modules. The sub modules can be web modules (WAR), ejb modules (JAR), resource adapter modules (RAR) or application client modules (`jar`). When an EAR consist of many sub modules, the deployment plans for all the sub modules can be provided in a single file named `geronimo-application.xml`. This single file contains the deployment details of each of the sub modules of the EAR. Alternatively, each of the sub modules can package its corresponding deployment plan file within itself. However, the preferable way is to provide a single deployment plan through `geronimo-application.xml` for all the sub modules. This mechanism provides flexibility of allowing us to modify the deployment configuration for all modules through a single file. In this section, we explore EAR deployment plan and understand what it contains.

There are 3 places a deployment plan (partial) can be associated with an ear, and they are used in this order:

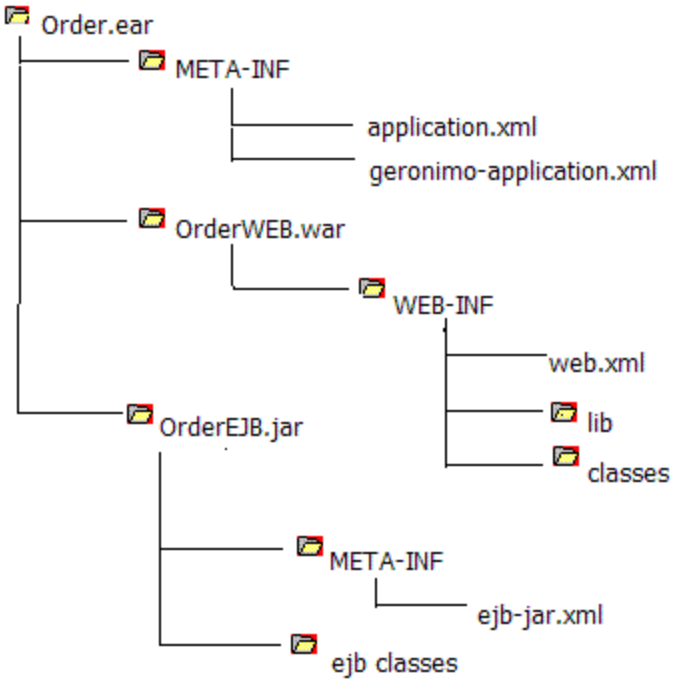
1. external plan to be specified at deployment time
2. ear level geronimo-application.xml
3. module level geronimo|openejb-*.xml

So, anything in an external plan takes precedence over bits in (2) or (3) configuring the same module. Anything in (2) takes precedence over a module level plan. If a module level plan is missing from (1) its looked for in (2),then (3); if missing from (1) and (2), then its looked for in the module (3).

There is no attempt to merge plans from these different locations: e.g. if there's something in (1) for a module, any other plans are ignored.

An enterprise application archive (EAR) should provide its deployment descriptor in the application.xml file. The `application.xml` lists all the sub modules in the EAR file along with the descriptions. In addition to the standard deployment descriptor, the EAR should also provide Geronimo specific deployment plan in geronimo-application.xml. Along with the description of each of the sub modules of the EAR file, this file also provides mappings for JEE resources that each of the sub modules refers in their deployment descriptor. The `geronimo-application.xml` is divided into several sections where in each section, the deployment plan for a sub module is provided. `geronimo-application.xml` is the highest level plan that provides deployment plan for all sub modules; hence it can contain XML elements from every other Geronimo XML schema used by Geronimo application deployer. The `geronimo-application.xml` is the super set of all other deployment plans.

For example, following is the structure of an EAR that has a web module and an ejb module.



The `Order.ear` file shown above contains two modules. One is `OrderWEB.war` file which is a web module and the other is `OrderEJB.jar` file which is an ejb module. The `META-INF` folder in `Order.ear` contains the application deployment descriptor (`application.xml`) and the Geronimo application deployment plan (`geronimo-application.xml`). The web application and the ejb application have packaged only their respective deployment descriptors. But the deployment plans for these modules are provided in the `geronimo-application.xml`.
The web application (`OrderWAR.war`) looks up stateless session bean in the `OrderEJB.jar` module to retrieve the order information. The `RetrieveOrderInfoBean` in `OrderEJB.jar` module uses JDBC connection to read the order information from a DB2 database.

The deployment descriptor of the OrderEJB.jar is as follows.

XMLsolidejb-jar.xml <?xml version="1.0" encoding="UTF-8" ?> <ejb-jar xmlns="http://java.sun.com/xml/ns/javaee" version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"> <description>Stateless Session Bean Example</description> <display-name>Stateless Session Bean Example</display-name> <enterprise-beans> <session> <ejb-name>RetrieveOrderInfoBean</ejb-name> <business-local> examples.session.stateless_dd.RetrieveOrderInfo </business-local> <ejb-class> examples.session.stateless_dd.RetrieveOrderInfoBean </ejb-class> <session-type>Stateless</session-type> <transaction-type>Container</transaction-type> <resource-ref> <res-ref-name>jdbc/DB2DataSource</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> <interceptors> <interceptor> <interceptor-class> examples.session.stateless_dd.RetrieveOrderCallbacks </interceptor-class> <post-construct> <lifecycle-callback-method>construct</lifecycle-callback-method> </post-construct> <pre-destroy> <lifecycle-callback-method>destroy</lifecycle-callback-method> </pre-destroy> </interceptor> </interceptors> <assembly-descriptor> <interceptor-binding> <ejb-name>RetrieveOrderInfoBean</ejb-name> <interceptor-class> examples.session.stateless_dd.RetrieveOrderCallbacks </interceptor-class> </interceptor-binding> </assembly-descriptor> </ejb-jar>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace.

In the `RetrieveOrderInfoBean`, the following code is used to look up the Datasource object and obtain a database connection.

JAVAsolidexamples.session.stateless_dd.RetrieveOrderInfoBean ... ... Context initContext = new InitialContext(); Context envContext = (Context) initContext.lookup("java:comp/env"); DataSource ds = (DataSource)envContext.lookup("jdbc/DB2DataSource"); System.out.println("Got DataSource\n"); con = ds.getConnection(); System.out.println("Got Connection\n"); ... ...

The deployment descriptor of the `OrderWEB.war` is as follows.

XMLsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5"> <display-name>OrderWEB</display-name> <welcome-file-list> <welcome-file>index.html</welcome-file> <welcome-file>index.htm</welcome-file> <welcome-file>index.jsp</welcome-file> <welcome-file>default.html</welcome-file> <welcome-file>default.htm</welcome-file> <welcome-file>default.jsp</welcome-file> </welcome-file-list> <servlet> <description></description> <display-name>RetrieveOrder</display-name> <servlet-name>RetrieveOrder</servlet-name> <servlet-class> examples.web.servlet.RetrieveOrder </servlet-class> </servlet> <ejb-local-ref> <ejb-ref-name>ejb/RetrieveOrderInfo</ejb-ref-name> <ejb-ref-type>Session</ejb-ref-type> <local> examples.session.stateless_dd.RetrieveOrderInfo </local> <ejb-link>RetrieveOrderInfoBean</ejb-link> </ejb-local-ref> <servlet-mapping> <servlet-name>RetrieveOrder</servlet-name> <url-pattern>/RetrieveOrder</url-pattern> </servlet-mapping> </web-app>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace.

In the `RetrieveOrder` servlet, the following code is used to look up the ejb to retrieve the order details.

JAVAsolidexamples.web.servlet.RetrieveOrder ... ... Context ctx = new InitialContext(); System.out.println("Instantiating beans..."); retrieveOInfo = (RetrieveOrderInfo)ctx.lookup("java:comp/env/ejb/RetrieveOrderInfo"); String orderIdStr = request.getParameter("orderid"); int orderId = Integer.parseInt (orderIdStr); OrderInfo oInfo = retrieveOInfo.getOrderInfo(orderId); ... ...

The deployment descriptor of the Order.ear is as follows.

XMLsolidapplication.xml <?xml version="1.0" encoding="UTF-8"?> <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:application="http://java.sun.com/xml/ns/javaee/application_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/j2ee/application_5.xsd" version="5"> <description>EAR Example</description> <display-name>Order Sample</display-name> <module> <web> <web-uri>OrderWEB.war</web-uri> <context-root>/OrderDemo</context-root> </web> </module> <module> <ejb>OrderEJB.jar</ejb> </module> </application>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace.

The deployment plan of the `Order.ear` is as follows.

XMLsolidgeronimo-application.xml <?xml version="1.0" encoding="UTF-8"?> <application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" application-name="Order"> <sys:environment> <sys:moduleId> <sys:groupId>Order</sys:groupId> <sys:artifactId>OrderEAR</sys:artifactId> <sys:version>5.0</sys:version> <sys:type>car</sys:type> </sys:moduleId> </sys:environment> <module> <web>OrderWEB.war</web> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1" > <sys:environment> <sys:moduleId> <sys:groupId>Order</sys:groupId> <sys:artifactId>OrderWEB</sys:artifactId> <sys:version>2.5</sys:version> <sys:type>war</sys:type> </sys:moduleId> </sys:environment> <context-root>/OrderDemo</context-root> </web-app> </module> <module> <ejb>OrderEJB.jar</ejb> <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>Order</sys:groupId> <sys:artifactId>OrderEJB</sys:artifactId> <sys:version>3.0</sys:version> <sys:type>jar</sys:type> </sys:moduleId> <sys:dependencies> <sys:dependency> <sys:groupId>console.dbpool</sys:groupId> <sys:artifactId>OrderDS</sys:artifactId> <sys:version>1.0</sys:version> <sys:type>rar</sys:type> </sys:dependency> </sys:dependencies> </sys:environment> <enterprise-beans> <session> <ejb-name>RetrieveOrderInfoBean</ejb-name> <naming:resource-ref> <naming:ref-name>jdbc/DB2DataSource</naming:ref-name> <naming:resource-link>OrderDS</naming:resource-link> </naming:resource-ref> </session> </enterprise-beans> </openejb-jar> </module> </application>
The default namespace of the above XML document is `http://geronimo.apache.org/xml/ns/j2ee/application-2.0`. The XML elements that do not have a namespace prefix belong to the default namespace.

Observe how the JEE 5 resource names and ejb names in `ejb-jar.xml` and `web.xml` are mapped to actual resources deployed in the server through `geronimo-application.xml`.

As we can observe from the `geronimo-application.xml`, the deployment plans for web and ejb modules are wrapped in `<module> .... </module>` elements. The xml elements used to provide deployment plan for the web module are from the schema http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1 which is the schema for `geronimo-web.xml`. Similarly, the xml elements used to provide ejb deployment plan are from the schema http://openejb.apache.org/xml/ns/openejb-jar-2.2 which is the schema for `openejb-jar.xml`. Hence, `geronimo-application.xml` borrows elements from all other schemas to provide deployment plan for its sub modules.

Also, observe that, in the `geronimo-application.xml`, along with moduleId configuration for the EAR itself, there is a moduleId configuration for each web and ejb modules. If the above EAR file deployed on the server and the configurations are listed, the following output would be displayed on the console.

The moduleId `Order/OrderEAR/5.0/car` is the configuration for the `Order.ear`. The ejb module declares a dependency on the `console.dbpool/OrderDS/1.0/rar` configuration in `<sys:dependencies>` section. This is the moduleId of the database pool that connects to the DB2 database where the order details are stored.

JEE Application Client deployment plan

JEE application client modules run in client container and also have access to server environment. Usually, JEE client applications are created to administer the running enterprise applications in the server. Client modules run in a separate JVM and connect to enterprise application resources but have access to all the application resources in standard JEE way.

The JEE client module requires application-client.xml as deployment descriptor and geronimo-application-client.xml as deployment plan. In the `application-client.xml`, the required ejb names, security role names, resources names etc., are declared while in `geronimo-application-client.xml`, the declared names are mapped to actual resources in server.

The following is the deployment descriptor of the JEE application client module that looks up an ejb and calls a method on it. The ejb converts the Indian Rupess (Rs.) into American Dollars ($). The client sends a double value which is Indian Rupees to ejb. The ejb returns equivalent American Dollars as double value.

XMLsolidapplication-client.xml <?xml version="1.0" encoding="UTF-8"?> <application-client xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application-client_5.xsd" version="5"> <ejb-ref> <ejb-ref-name>ejb/Converter</ejb-ref-name> <ejb-ref-type>Session</ejb-ref-type> <remote>examples.appclient.Converter</remote> </ejb-ref> </application-client>
The default namespace of the above XML document is `http://java.sun.com/xml/ns/javaee`. The XML elements that do not have a namespace prefix belong to the default namespace. Hence, in the above XML document, all the XML elements belong to the default namespace.

The application client declares the ejb name `ejb/Converter` through `<ejb-ref>>` .. `</ejb-ref>` elements.

Following is the corresponding deployment plan of the JEE client module.

XMLsolidgeronimo-application-client.xml <?xml version="1.0" encoding="UTF-8"?> <application-client xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:security="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:connector="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <sys:client-environment> <sys:moduleId> <sys:groupId>Converter</sys:groupId> <sys:artifactId>Converter-app-client</sys:artifactId> <sys:version>3.0</sys:version> <sys:type>jar</sys:type> </sys:moduleId> </sys:client-environment> <sys:server-environment> <sys:moduleId> <sys:groupId>Converter</sys:groupId> <sys:artifactId>Converter-app-client-server</sys:artifactId> <sys:version>3.0</sys:version> <sys:type>jar</sys:type> </sys:moduleId> </sys:

server-environment> <ejb-ref> <ref-name>ejb/Converter</ref-name> <naming:pattern> <naming:groupId>Converter</naming:groupId> <naming:artifactId>ConverterEAR</naming:artifactId> <naming:version>5.0</naming:version> <naming:module>ConverterEJB.jar</naming:module> <naming:name>ConverterBean</naming:name> </naming:pattern> </ejb-ref> </application-client>
The default namespace of the above XML document is `http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0`. The XML elements that do not have a namespace prefix belong to the default namespace. Hence, in the above XML document, `<application-client>`, `<ejb-ref>` and `<ref-name>` elements belong to the default namespace.

Observe the various xml elements and schemas to which they belong. The plan defines the client environment and the server environment configurations. The server environment configuration runs in the server where as the client environment configuration runs in the client JVM. In the above plan, the ejb name `ejb/Converter` is mapped to `ConverterBean` ejb in the `ConverterEAR`.

The below is the client code that looks up the ejb and calls the method on it.

JAVAsolidConverterClient.java package examples.appclient.client; import javax.naming.Context; import javax.naming.InitialContext; import examples.appclient.Converter; public class ConverterClient { //The remote interface of the ConverterBean packaged with the //JEE client jar private static Converter converter; private static double amount = 50; public static void main(String[] args) { amount = Double.parseDouble(args[0]); doConversion(); } public static void doConversion() { try { Context context = new InitialContext(); converter = (Converter) context.lookup("java:comp/env/ejb/Converter"); double dollars = converter.getDollars(amount); System.out.println("Rs " + amount + " is " + dollars + " Dollars."); System.exit(0); } catch (Exception ex) { System.err.println ("Caught an unexpected exception!"); ex.printStackTrace(); } } }

The `META-INF/MANIFEST.MF` file should contain the following entry for the client to run.

XMLsolidMANIFEST.MF Manifest-Version: 1.0 Main-Class: examples.appclient.client.ConverterClient
Do not forget to insert a new line after the Main-Class: entry in the MANIFEST.MF file.

The JEE client is created by packaging `META-INF/application-client.xml`, `META-INF/geronimo-application-client.xml`, `ConverterClient.class`, `Converter.class` and `META-INF/MANIFEST.MF` files into a jar file.

The following command illustrates the packaging.

The following commands illustrates the deployment and running of the client module.

Message Driven Bean deployment plan.

Apache geronimo ships with ActiveMQ message broker and an inbound and outbound JMS resource adapter for the ActiveMQ broker. This sample illustrates deploying and running two MDBs that listen to a jms topic `TextTopic`. When the web client publishes a message to this topic, the two MDBs receive the message and process it. Because the message is published to the topic, all the configured listeners, in this case the two MDBs, receive a copy of the message. In addition to that, we will also illustrate how to deploy a JMS resource adapter within the application scope. Usually, the resource adapters are deployed at the server scope and can be used by all other applications as well. In this example, a JMS resource plan is embedded in the application deployment plan `geronimo-application.xml` and deployed while deploying the application archive.

XMLsolidejb-jar.xml <?xml version="1.0" encoding="UTF-8" ?> <ejb-jar> <enterprise-beans> <message-driven> <ejb-name>TextMessageBean1</ejb-name> <ejb-class> sample.mdb.TextMessageBean1 </ejb-class> <messaging-type> javax.jms.MessageListener </messaging-type> <transaction-type>Bean</transaction-type> <message-destination-type> javax.jms.Topic </message-destination-type> <activation-config> <activation-config-property> <activation-config-property-name> destinationType </activation-config-property-name> <activation-config-property-value> javax.jms.Topic </activation-config-property-value> </activation-config-property> </activation-config> </message-driven> <message-driven> <ejb-name>TextMessageBean2</ejb-name> <ejb-class> sample.mdb.TextMessageBean2 </ejb-class> <messaging-type> javax.jms.MessageListener </messaging-type> <transaction-type>Bean</transaction-type> <message-destination-type> javax.jms.Topic </message-destination-type> <activation-config> <activation-config-property> <activation-config-property-name> destinationType </activation-config-property-name> <activation-config-property-value> javax.jms.Topic </activation-config-property-value> </activation-config-property> </activation-config> </message-driven> </enterprise-beans> </ejb-jar>

The `ejb-jar.xml` declares the two MDBs `sample.mdb.TextMessageBean1` and `sample.mdb.TextMessageBean2` both listen to a javax.jms.Topic destination.

XMLsolidopenejb-jar.xml <?xml version="1.0" encoding="UTF-8"?> <openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>Sample</sys:groupId> <sys:artifactId>MDB-EJB</sys:artifactId> <sys:version>1.0</sys:version> <sys:type>car</sys:type> </sys:moduleId> </sys:environment> <enterprise-beans> <message-driven> <ejb-name> TextMessageBean1 </ejb-name> <nam:resource-adapter> <nam:resource-link> TradeJMSResources </nam:resource-link> </nam:resource-adapter> <activation-config> <activation-config-property> <activation-config-property-name> destination </activation-config-property-name> <activation-config-property-value> TextMessageTopic </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.Topic</activation-config-property-value> </activation-config-property> </activation-config> </message-driven> <message-driven> <ejb-name>TextMessageBean2</ejb-name> <nam:resource-adapter> <nam:resource-link> TradeJMSResources </nam:resource-link> </nam:resource-adapter> <activation-config> <activation-config-property> <activation-config-property-name> destination </activation-config-property-name> <activation-config-property-value> TextMessageTopic </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.Topic </activation-config-property-value> </activation-config-property> </activation-config> </message-driven> </enterprise-beans> </openejb-jar>

In the deployment plan `openejb-jar.xml`, the two MDBs are configured as end point listeners for the jms topic `TextMessageTopic`.

The code for the two MDBs are as follows.

JAVAsolidTextMessageBean1.java package sample.mdb; import javax.jms.JMSException; import javax.jms.Message; import javax.jms.TextMessage; public class TextMessageBean1 { public TextMessageBean1() { } public void onMessage(Message msg) { if (msg instanceof TextMessage) { TextMessage tm = (TextMessage) msg; try { String text = tm.getText(); System.out.println("Received new message in TextMessageBean1 : " + text); System.out.println ("CustomerId : " + tm.getIntProperty("CustomerId")); System.out.println("CustomerName : " + tm.getStringProperty("CustomerName")); } catch (JMSException e) { e.printStackTrace(); } } } } JAVAsolidTextMessageBean2.java package sample.mdb; import javax.jms.JMSException; import javax.jms. Message; import javax.jms.TextMessage; public class TextMessageBean2 { public TextMessageBean2() { } public void onMessage(Message msg) { if (msg instanceof TextMessage) { TextMessage tm = (TextMessage) msg; try { String text = tm.getText(); System.out.println("Received new message in TextMessageBean2 : " + text); System.out.println("CustomerId : " + tm.getIntProperty("CustomerId")); System.out.println("CustomerName : " + tm. getStringProperty("CustomerName")); } catch (JMSException e) { e.printStackTrace(); } } } }

After receiving the message, the MDBs just print the contents of the message.

The web client for the application is as follows.

XMLsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun. com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java. sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5"> <display-name>MDBSampleWEB</display-name> <welcome-file-list> <welcome-file>index.html</welcome-file> <welcome-file>index.htm</welcome-file> <welcome-file>index.jsp</welcome-file> </welcome-file-list> <servlet> <description></description> <display-name>Test</display-name> <servlet-name>Test</servlet-name> <servlet-class>sample.mdb.Test</servlet-class> < /servlet> <resource-ref> <description>jms broker</description> <res-ref-name>jms/broker</res-ref-name> <res-type> javax.jms.TopicConnectionFactory < /res-type> <res-auth>Container</res-auth> </resource-ref> <resource-env-ref> <description>Predefined Topic</description> <resource-env-ref-name> jms /Topic/TextTopic </resource-env-ref-name> <resource-env-ref-type> javax.jms.Topic</resource-env-ref-type> </resource-env-ref> <servlet-mapping> <servlet-name>Test</servlet-name> <url-pattern>/Test</url-pattern> </servlet-mapping> </web-app>

The web client declares the `javax.jms.TopicConnectionFactory` and the topic to which the servlet has to publish the message. The names declared here are mapped to actual resources in the `geronimo-web.xml` as follows.

XMLsolidgeronimo-web.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1" xmlns:nam=" http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns /deployment-1.2"> <sys:environment> <sys:moduleId> <sys:groupId>sample</sys:groupId> <sys:artifactId>MDB-Web</sys:artifactId> <sys:version>1.0< /sys:version> <sys:type>car</sys:type> </sys:moduleId> </sys:environment> <context-root>/MDBSampleWEB</context-root> <nam:resource-ref> <nam: ref-name> jms/broker</nam:ref-name> <nam:resource-link> jms/TopicConnectionFactory </nam:resource-link> </nam:resource-ref> <nam:resource-env-ref> <nam:ref-name> jms/Topic/TextTopic </nam:ref-name> <nam:message-destination-link> TextMessageTopic </nam:message-destination-link> </nam: resource-env-ref> </web-app>

Please note that the `jms/TopicConnectionFactory` and `jms/Topic/TextTopic` are the names of the actual connection factory and jms topic. These are deployed by a jms resource plan embedded in the EAR's deployment plan as follows.

XMLsolidapplication.xml <?xml version="1.0" encoding="ASCII"?> <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=http://java.sun.com/xml/ns/javaee xmlns:app="http://java.sun.com/xml/ns/javaee/application_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns /javaee http://java.sun.com/xml/ns/javaee/application_5.xsd" version="5"> <display-name>MDBSampleEAR</display-name> <module> <ejb>MDBSampleEJB.jar</ejb> </module> <module> <web> <web-uri>MDBSampleWEB.war</web-uri> <context-root>MDBSampleWEB</context-root> < /web> </module> </application> XMLsolidgeronimo-application.xml <?xml version="1.0" encoding="UTF-8"?> <application xmlns="http://geronimo.apache. org/xml/ns/j2ee/application-2.0" xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2" application-name="MDBSampleEAR"> <sys:environment> <sys:moduleId> <sys:groupId>default</sys:groupId> <sys:artifactId>MDBSampleEAR</sys:artifactId> <sys:version>1.0</sys:version> <sys:type>car</sys: type> </sys:moduleId> </sys:environment> <ext-module> <connector>TopicJMSSample</connector> <external-path> <sys:groupId> org.apache. geronimo.modules </sys:groupId> <sys:artifactId> geronimo-activemq-ra </sys:artifactId> <sys:version>2.1</sys:version> </external-path> <connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"> <resourceadapter> <!--how to connect to the JMS Server--> <resourceadapter-instance> <resourceadapter-name> TradeJMSResources </resourceadapter-name> <config-property-setting name="ServerUrl"> tcp://localhost:61616 </config-property-setting> <config-property-setting name="UserName"> not needed </config-property-setting> <config-property-setting name="Password"> not needed </config-property-setting> <workmanager> <gbean-link>DefaultWorkManager</gbean-link> </workmanager> </resourceadapter-instance> <!-- defines a ConnectionFactory--> <outbound-resourceadapter> <connection-definition> <connectionfactory-interface> javax.jms.ConnectionFactory < /connectionfactory-interface> <connectiondefinition-instance> <name>jms/TopicConnectionFactory</name> <implemented-interface> javax.jms. TopicConnectionFactory </implemented-interface> <connectionmanager> <xa-transaction> <transaction-caching/> </xa-transaction> <single-pool> <max-size>10</max-size> <min-size>0</min-size> <blocking-timeout-milliseconds> 5000 </blocking-timeout-milliseconds> <idle-timeout-minutes>0</idle-timeout-minutes> <match-one/> </single-pool> </connectionmanager> </connectiondefinition-instance> </connection-definition> </outbound-resourceadapter> </resourceadapter> <adminobject> <adminobject-interface> javax.jms.Topic</adminobject-interface> <adminobject-class> org.activemq. message.ActiveMQTopic </adminobject-class> <adminobject-instance> <message-destination-name> TextMessageTopic </message-destination-name> <config-property-setting name="PhysicalName"> TextMessageTopic </config-property-setting> </adminobject-instance> </adminobject> </connector> < /ext-module> </application>

The plan for resource adapter is provided under `<ext-module>` xml elements. Also, the resource adapter archive `(rar)` file to be used to deploy the plan is mentioned using `external-path` xml element; the resource adapter plan follows these elements. The plan deploys{{ jms /TopicConnectionFactory}} and `TextTopic`.

The reference to resource archive file is provided using the following xml elements in the above plan.

JAVAsolidReferencing Libraries <external-path> <sys:groupId> org.apache.geronimo.modules </sys:groupId> <sys:artifactId> geronimo-activemq-ra </sys: artifactId> <sys:version>2.1</sys:version> </external-path>

The above pattern is the way how geronimo references various libraries available in the server repository. Any external libraries can also be uploaded to server repository using admin console portlet. After starting the server, click on `Console Navigation => Services => Repository` to display `Repo sitory Viewer` portlet. In this portlet, users can upload required third party libraries by providing proper `groupId, artifactId and version` values for the libraries being uploaded. The activeMQ rar file is also available in the repository as `org.apache.geronimo.modules/geronimo-activemq-ra/2.1/rar`. Click on this link to get the usage instructions on how to reference this library in other modules.

The web client that look up the connection factory and topic and sends messages is as follows.

JAVAsolidTest.java package sample.mdb; import java.io.IOException; import java.io.PrintWriter; import javax.jms.DeliveryMode; import javax.jms.Session; import javax.jms.TextMessage; import javax.jms.Topic; import javax.jms.TopicConnection; import javax.jms.TopicConnectionFactory; import javax.jms.TopicPublisher; import javax.jms.TopicSession; import javax.naming.InitialContext; import javax.servlet.ServletException; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import javax.jms.Connection; public class Test extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet { static final long serialVersionUID = 1L; private TopicConnection connection; private Topic topic; public Test() { super(); } public void init() throws ServletException { String connectionFactoryName = "java:comp/env/jms/broker"; String topicName = "java:comp/env/jms/Topic/TextTopic"; try { InitialContext naming = new InitialContext(); TopicConnectionFactory connectionFactory = (TopicConnectionFactory) naming.lookup(connectionFactoryName); connection = connectionFactory.createTopicConnection(); topic = (Topic) naming.lookup(topicName); }catch(Exception e) { e.printStackTrace(); throw new ServletException(e); } } public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { TopicSession publishSession = null; PrintWriter out = response.getWriter(); try { boolean transacted = false; publishSession = connection.createTopicSession(transacted, Session.AUTO_ACKNOWLEDGE); TopicPublisher sender = publishSession.createPublisher(topic); sender.setDeliveryMode(DeliveryMode.PERSISTENT); TextMessage message = publishSession.createTextMessage("Customer Info"); message.setIntProperty("CustomerId", Integer.parseInt(request.getParameter("CustomerId"))); message.setStringProperty("CustomerName", request.getParameter("CustomerName")); sender.send(message); out.println("Message is successfully sent...!!"); } catch(Exception e) { throw new ServletException(e); } finally { try { if (publishSession != null) { publishSession.close(); } } catch (Exception e) {} } } protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { } public void destroy() { if (connection != null) { try { if (connection != null) { connection.close(); } } catch (Exception e) { } } } }

When the above servlet is accessed on a browser window as `http://localhost:8080/MDBSampleWEB/Test?CustomerId=10&CustomerName=Phani`, the following output is displayed in the server console.