

Migrating from Apache Tomcat Version 6.0.x to WebSphere Application Server Community Edition V2.1

Skill Level: Intermediate

[Ashish Jain](#)
Software Engineer
IBM

06 Aug 2008

This article walks you through deploying an application to Tomcat 6.0, then migrating the code and deploying it to WebSphere® Application Server Community Edition V2.1. The sample application highlights some of the notable differences between the two implementations.

Introduction

Download Community Edition V2.1 now!

IBM WebSphere Application Server Community Edition V2.1 is available for no charge to use and deploy. [Download it now](#) to get started.

IBM WebSphere Application Server Community Edition (hereafter referred to as Community Edition) is a Java™ Enterprise Edition 5 (Java EE5) certified application server that contains integrated components for data (Apache Derby), messaging services (Active MQ), Web services (Apache Axis), etc.

It is built using the Geronimo Beans (GBean) Architecture with Apache Geronimo as its core. Its small footprint, ease of use, free availability and IBM world-class, optional support makes it ideal for small and medium sized organizations.

Apache Tomcat is a Web container developed by the Apache Software Foundation. It implements the Java Servlet and Java Server Pages API. Tomcat Version 6.0.x implements Java Servlet 2.5 and Java Server Pages 2.1 APIs.

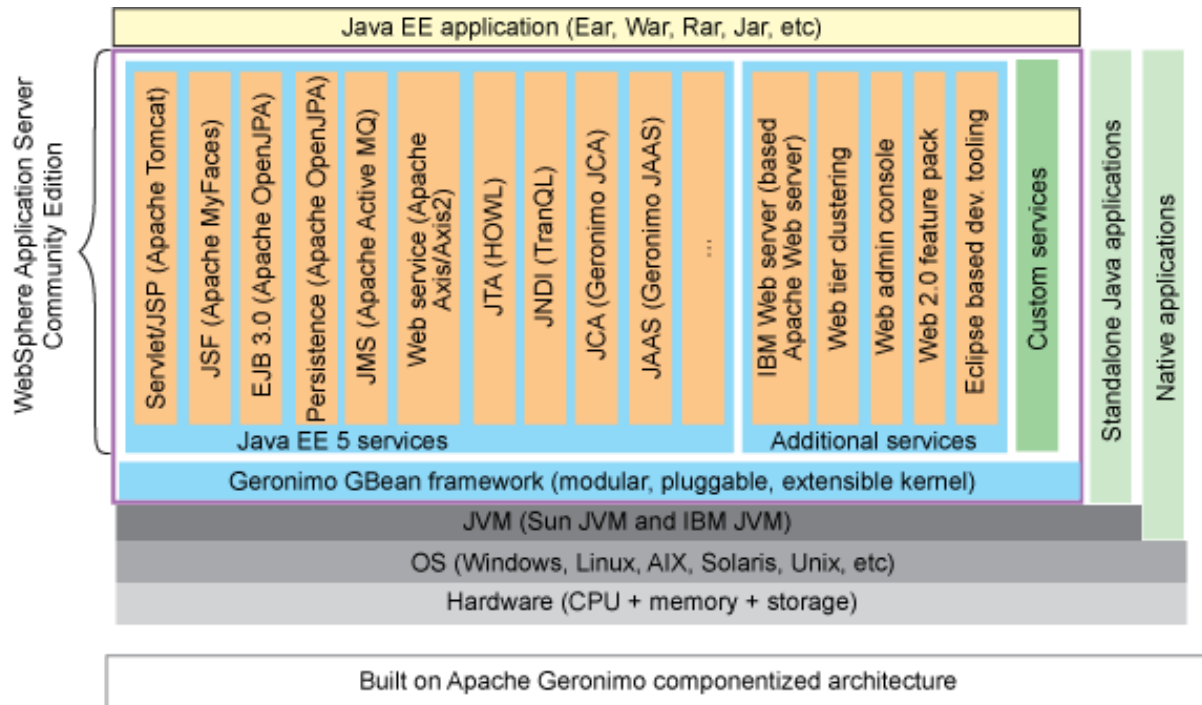
While this article will take you through migration of a sample application from Tomcat 6.0.x to Community Edition 2.1, migrating from Tomcat 5.x would be very similar. Tomcat Version 5.x implements Java Servlet 2.4 and Java Server Pages 2.0 APIs, and Community Edition 2.1 is backward compatible to these versions. However, there may be some differences in how you define the configurations.

This article walks you through the migration of an application developed on Apache Tomcat to Community Edition. We have used the Eclipse IDE for application development on Tomcat and Community Edition. The article is organized by the following sections:

- [Architectural analysis - similarities and differences](#)
- [About the sample application](#)
- [Setting up Tomcat for application deployment](#)
- [Deploying the application in Tomcat](#)
- [Migrating the application to Community Edition](#)
- [Removing unneeded services from the Community Edition WAR file](#)
- [Deploying the application in Community Edition](#)
- [Troubleshooting migration problems](#)

Architectural analysis - similarities and differences

Figure 1. Geronimo (Community Edition) architecture



As Figure 1 suggests, Community Edition follows a component architecture. It comprises best-of-breed, open-source products from various open source communities. Apache Tomcat is the default Web container for Community Edition, so you can easily migrate an application developed on Tomcat to Community Edition.

While working with Community Edition you will be working with the same Tomcat server as the standalone version of Tomcat. As a result you don't need to repackage the application for Community Edition. However there are few differences you need to work out while migrating the application from Tomcat to Community Edition. These differences arise because Tomcat is deployed as a service within Community Edition to provide the features available for a Java EE5 certified application server; not just a servlet container. Tomcat in Community Edition is integrated with other deployed container services such as:

- JMS (Apache ActiveMQ)
- Database (Apache Derby)
- EJB (Apache OpenEJB)
- Web services (Apache Axis)
- Persistence (Apache OpenJPA)
- Logging
- Security

To overcome most migration issues, you need to figure out how each service maps within Tomcat and applies to Community Edition. The basic architectural of Tomcat is the same, but in Community Edition it is defined as a gbean. This implementation allows newer versions of Tomcat to be easily integrated into upcoming versions of Community Edition

The following table illustrates the major differences and similarity between standalone Tomcat versus Tomcat running in Community Edition:

Table 1. Tomcat V6.0.x and Community Edition V2.1 feature comparison

Feature	Tomcat	Community Edition
Java Servlet	Implements Servlet 2.5 API	Implements Servlet 2.5 API
Java Server Pages	Implements JSP 2.1 API	Implements JSP 2.1 API
JNDI	Manual Configuration usually done by <code>server.xml</code>	JNDI has access to object references managed by Geronimo kernel
JMS	Manual configuration done in <code>context.xml</code>	Web-based GUI can be used to create JMS resources
JDBC data source	Manual configuration done in <code>context.xml</code>	Web-based GUI can be used to create a JDBC datasource
Web service (Apache Axis)	Deployment of Axis.war introduces WebServices functionality	Axis is pre-integrated onto Community Edition.
Security realms	Configured using <code>server.xml</code> or <code>context.xml</code> files	Supports container-managed realms, which are available for authorization of all components. Individual component realms may also be defined.
Multiple connectors	Multiple protocol handlers to access the same engine. An HTTP connector can work with an SSL connector on the same engine	Administrative console can be used to configure various connectors including HTTP, SSL, and Apache JServ protocol (AJP13) used for load balancing and clustering
Virtual host	Enables hosting of different Web sites on same IP address. This can be configured using <code>server.xml</code>	Similar feature can be achieved by configuring <code>config.xml</code>
Deployment descriptor	<code>web.xml</code> is the default deployment descriptor for a J2EE Web application (WAR)	<code>web.xml</code> is the default deployment descriptor for a J2EE Web application (WAR)
Deployment plan	No vendor-specific deployment descriptor	Community Edition has Geronimo-specific deployment descriptor to configure Geronimo specific services. For a Web application, <code>geronimo-web.xml</code> is the

deployment plan.

In Community Edition, most of the configuration can be done using one of these deployment plans: `geronimo-web.xml`, `config.xml` or `config-substitution.properties`. You can also easily add resources like JMS queues, connection factories, and JDBC data sources using the Web-based administrative console. The console simplifies the configuration management versus the manual configuration done using `server.xml` and `context.xml` in Tomcat.

About the sample application

The sample application we migrated from Tomcat V6.0.x to Community Edition V2.1 is a general store application that uses the following components:

- JSPs
- Servlets
- JSP Standard Tag Library (JSTL)
- Apache Derby JDBC data source
- Apache ActiveMQ-based JMS
- Apache Axis-based Web service

The application also uses the default Tomcat `UserRealm` to authenticate users, whereas Community Edition uses the default `geronimo-admin` realm.

The sample application included with this article consists of a single WAR module with the following content:

- **generalstore.jsp** - Presents the shopping catalog to the user. Displays current IBM stock price through an external web service
- **checkoutcart.jsp** - Presents the shopping cart to the user. Uses RDBMS through a Derby data source to display the promotional messages. Displays the IBM stock price through an external Web service.
- **generalstore.css** - Stylesheet used to format HTML elements by both JSP pages.
- **StockService.wsdl** - Describes the location of Web services and methods available within the services. You use this WSDL file to generate the required Java classes, and Eclipse tooling to generate these class files.

- **StoreController.java** - Main controller for dispatching requests to JSPs. Also obtains category and product information and attaches them as attributes for JSPs to display.
- **GeneralStore.java** - Contains methods that return the data required for the application. Also contains functions that implement a custom tag library (see `generalstore-taglib.tld`).
- **Category.java, Product.java, Lineltem.java** - JavaBeans used to hold values when data is transferred between the model and view of the application.
- **CheckOut.java** - Completes the order and sends it to the Inventory manager

Setting up Tomcat for application deployment

Tomcat connector configuration

By default the Tomcat server listens for a request on port 8080. To change the default port, you modify `<Tomcat_Home>/conf/server.xml`. Listing 1 shows the modification required to change the default port configuration.

Listing 1. Tomcat connector configuration in server.xml

```
<Service name="Catalina">
  ...
  ...
  <Connector port="8080"
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

Change the port value to anything you like. Please remember to stop the server before making any configuration changes to `server.xml`. Restart the server for the settings to take effect.

Tomcat authentication realm configuration

A realm maintains user, group, and password information in Tomcat. By default, you configure a `UserDatabaseRealm` in Tomcat, enabling you to authenticate a user before letting them access the store. The `server.xml` file also sets up this realm. The bold lines in the `server.xml` code segment in Listing 2 represent the configuration for this authentication realm.

Listing 2. Tomcat UserDatabaseRealm configuration in server.xml

```

<Server port="8005" shutdown="SHUTDOWN">
  ...
  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
      type="org.apache.catalina.UserDatabase"
      description="User database that can be updated and saved"
      factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
      pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>
  ...
  <Service name="Catalina">

    <Connector port="8080" protocol="HTTP/1.1"
      connectionTimeout="20000"
      redirectPort="8443" />

    ...

    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
      resourceName="UserDatabase" />
  </Service>
</Server>

```

The above configuration sets up the `UserDatabaseRealm` to use a file under `conf/tomcat-users.xml` to maintain user, password, and role information. Because this realm is configured at the `<Engine>` level, it is available to all the hosts that are running on the Tomcat `<Engine>` component instance.

To run our application, you do not have to make any modifications to the `conf/server.xml` file, since the default configuration will work for us as is. You do, however, need to make some changes to the `conf/tomcat-users.xml` file, where the `UserDatabaseRealm` keeps all the user information. Make the change indicated in bold in Listing 3 to the `tomcat-users.xml` file.

Listing 3. Adding admin and manager roles for the Tomcat user in `tomcat-users.xml`

```

<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat,manager,admin"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>

```

This code assigns the `tomcat` user (with password 'tomcat') roles as part of the manager and admin groups.

In the application's `web.xml` file, the store's controller is protected and grants access only to users with an `###admin###` role. The bold lines in Listing 4 show the `web.xml` code (in the `dd` directory of the code distribution), that protects the application.

Listing 4. `web.xml` deployment descriptor with authenticated access

protection for the application

```

<?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>ShoppingStore</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>
    <servlet-name>GeneralStore</servlet-name>
    <servlet-class>com.ibm.wasce.store.StoreController</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>GeneralStore</servlet-name>
    <url-pattern>/store.cgi</url-pattern>
  </servlet-mapping>
  <resource-ref>
    <description>Derby DB connection</description>
    <res-ref-name>jdbc/storeDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
  <security-constraint>
    <display-name>General Store Security Constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Entire store</web-resource-name>
      <url-pattern>*.cgi</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Shopping Realm</realm-name>
  </login-config>
  <security-role>
    <role-name>admin</role-name>
  </security-role>
  <servlet>
    <description></description>
    <display-name>CheckOut</display-name>
    <servlet-name>CheckOut</servlet-name>
    <servlet-class>com.ibm.wasce.store.CheckOut</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CheckOut</servlet-name>
    <url-pattern>/CheckOut</url-pattern>
  </servlet-mapping>
</web-app>

```

The `<security-constraint>` and `<security-role>` elements in the code above are standard J2EE security configuration elements, and are portable between containers. As such, the same unchanged `web.xml` will work later in the Community

Edition environment.

Setting up the network Derby server

First make sure you have Apache Derby 10.3.1 installed. Then follow these steps:

1. Set the `DERBY_INSTALL` environment variable to your Derby installation directory. Include the following elements in your `CLASSPATH` environment variable:
 - `lib/derby.jar`
 - `lib/derbytools.jar`
 - `lib/derbynet.jar`
 - `lib/derbyclient.jar`
2. Start the Derby server using this command:

```
java org.apache.derby.drda.NetworkServerControl start
```

The Derby server starts and listens at the default port 1527.

3. Create a database named `generalstore`, with only one table called `promotions`. The `createdb1.sql` script creates and populates the database.
Use the `ij` command to execute the SQL script:

```
java org.apache.derby.tools.ij createdb1.sql
```

The `createdb1.sql` script contains the SQL commands shown in Listing 5.

Listing 5. The `createdb1.sql` script: Creates database table and fills with data

```
connect 'jdbc:derby://localhost/generalstore;create=true';
drop table promotion;
create table promotion
  (id char (5) not null,
   message char(40) not null,
   primary key(id));
insert into promotion values ('1', 'Thank your for your order.');
```

If you see the message `ERROR 08001: No suitable driver`, then the required `lib/derbyclient.jar` library is not in your `CLASSPATH` environment variable.

Setting up the Apache ActiveMQ JMS Broker

Make sure you have `apache-activemq-4.1.1` installed and then follow these steps:

1. Include the following elements in your CLASSPATH environment variable.
 - `lib/activeio-core-3.0.0-incubator.jar`
 - `lib/commons-logging-1.1.jar`
 - `<ActiveMQ_HOME>/apache-activemq-4.1.1.jar`
2. Start the ActiveMQ broker by running the `activemq.bat` script. You can find this script in `<ActiveMQ_HOME>/bin`

Setting up Apache Axis

Include the following elements in your CLASSPATH environment variable. You can automatically download the JARs mentioned below from within Eclipse. See [Removing unneeded services from the Community Edition WAR file](#) for more information.

- `wSDL4j-1.5.1.jar`
- `saaj.jar`
- `jaxrpc.jar`
- `axis.jar`

Tomcat JNDI resource reference and JDBC connector

The `checkoutcart.jsp` page uses JSTL's SQL support tags to access the promotion database, and prints out the promotional messages. Listing 6 shows the code excerpt that accesses the RDBMS.

Listing 6. JSP code in `checkoutcart.jsp` that access the database server

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="storetags" uri="/GeneralStoreTagLibrary" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
...
<sql:query var="rs" dataSource="jdbc/storeDB">
  select * from promotion
</sql:query>
...
<tr>
<td colspan="5">
<c:forEach var="row" items="{rs.rows}">
```

```

    ${row.message}<br/>
  </c:forEach>
</td>
</tr>

```

The JSTL `<sql:query>` tag will look up, via JNDI, the data source named `jdbc/storeDB`. This name reference is defined in a J2EE standard manner within the `web.xml` deployment descriptor of the application. This standard configuration will work with both Tomcat and Community Edition. Listing 7 shows this configuration.

Listing 7. Element in web.xml specifying the JNDI datasource reference

```

<resource-ref>
  <description>Derby DB connection</description>
  <res-ref-name>jdbc/storeDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

```

You now need to configure this reference to match the actual JDBC driver and RDBMS connection, in a container-specific manner (different for Tomcat and Community Edition). For Tomcat, you configure it in the `conf/context.xml` file (found in the Tomcat Installation directory). See Listing 8:

Listing 8. Tomcat specific context.xml descriptor associating the JNDI resource with the Derby JDBC connector

```

<Context>
  ...
  ...
  <Resource name="jdbc/storeDB" auth="Container" type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    user="APP" password="APP"
    url="jdbc:derby://localhost/generalstore"/>
  ...
  ...
</Context>

```

Configuring Tomcat for ActiveMQ broker

The `Checkout.java` servlet displays a message when the order is successfully placed. Listing 9 shows the servlet code excerpt, which looks up the connection factory and queue:

Listing 9. Servlet code excerpt

```

Context initContext = new InitialContext();
Context jndiContext=(Context) initContext.lookup("java:comp/env");

```

```
connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/ConnectionFactory");
queue = (Queue) jndiContext.lookup("jms/aQueue");
```

You define the name reference to the connection factory and queue in web.xml, as shown in listing 10.

Listing 10. Elements in web.xml specifying JMS connection factory and queue

```
<resource-ref>
  <description>jms broker</description>
  <res-ref-name>jms/ConnectionFactory</res-ref-name>
  <res-type>javax.jms.ConnectionFactory</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
<message-destination-ref>
  <message-destination-ref-name>jms/aQueue</message-destination-ref-name>
  <message-destination-type>javax.jms.Queue</message-destination-type>
  <message-destination-usage>ConsumesProduces</message-destination-usage>
  <message-destination-link>aQueue</message-destination-link>
</message-destination-ref>
<message-destination>
  <message-destination-name>aQueue</message-destination-name>
</message-destination>
```

You now need to configure Tomcat for an ActiveMQ connection factory and destination queue, also in a container-specific manner. For Tomcat, you can configure it in the `conf/context.xml` file (found in the Tomcat Installation directory). Listing 11 shows the connection factory and destination queue parameters.

Listing 11. Tomcat-specific context.xml descriptor for ActiveMQ broker configuration

```
<Context>
  ...
  ...
  <Resource
    name="jms/ConnectionFactory"
    auth="Container"
    type="org.apache.activemq.ActiveMQConnectionFactory"
    description="JMS Connection Factory"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    brokerURL="tcp://localhost:61616"
    brokerName="localhost"
    persistent="false"
    useEmbeddedBroker="false" />

  <Resource name="jms/aQueue"
    auth="Container"
    type="org.apache.activemq.command.ActiveMQQueue"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    physicalName="MY.TEST.QUEUE" />
  ...
  ...
</Context>
```

Generating the Web services class using StockService.wsdl

In this section we generate the Web services class:

1. Listing 12 shows the WSDL file needed to generate the Web services artifacts.

Listing 12. StockService.wsdl for generating Web services artifacts

```
<wsdl:definitions xmlns:http=http://schemas.xmlsoap.org/wsdl/http/
  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns:s=http://www.w3.org/2001/XMLSchema
  xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:tns=http://www.webserviceX.NET/
  xmlns:tm=http://microsoft.com/wsdl/mime/textMatching/
  xmlns:mime=http://schemas.xmlsoap.org/wsdl/mime/
  targetNamespace=http://www.webserviceX.NET/
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://www.webserviceX.NET/">
      <s:element name="GetQuote">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="symbol"
              type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetQuoteResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetQuoteResult" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string"/>
    </s:schema>
  </wsdl:types>

  <wsdl:message name="GetQuoteSoapIn">
    <wsdl:part name="parameters" element="tns:GetQuote"/>
  </wsdl:message>

  <wsdl:message name="GetQuoteSoapOut">
    <wsdl:part name="parameters" element="tns:GetQuoteResponse"/>
  </wsdl:message>

  <wsdl:message name="GetQuoteHttpGetIn">
    <wsdl:part name="symbol" type="s:string"/>
  </wsdl:message>

  <wsdl:message name="GetQuoteHttpGetOut">
    <wsdl:part name="Body" element="tns:string"/>
  </wsdl:message>

  <wsdl:message name="GetQuoteHttpPostIn">
    <wsdl:part name="symbol" type="s:string"/>
  </wsdl:message>

  <wsdl:message name="GetQuoteHttpPostOut">
    <wsdl:part name="Body" element="tns:string"/>
  </wsdl:message>

  <wsdl:portType name="StockQuoteSoap">
    <wsdl:operation name="GetQuote">
      <wsdl:input message="tns:GetQuoteSoapIn"/>
      <wsdl:output message="tns:GetQuoteSoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
```

```

<wsdl:portType name="StockQuoteHttpGet">
  <wsdl:operation name="GetQuote">
    <wsdl:input message="tns:GetQuoteHttpGetIn"/>
    <wsdl:output message="tns:GetQuoteHttpGetOut"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:portType name="StockQuoteHttpPost">
  <wsdl:operation name="GetQuote">
    <wsdl:input message="tns:GetQuoteHttpPostIn"/>
    <wsdl:output message="tns:GetQuoteHttpPostOut"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="StockQuoteSoap" type="tns:StockQuoteSoap">
  <soap:binding transport=http://schemas.xmlsoap.org/soap/http
    style="document"/>
  <wsdl:operation name="GetQuote">
    <soap:operation soapAction=http://www.webserviceX.NET/GetQuote
      style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="StockQuoteHttpGet" type="tns:StockQuoteHttpGet">
  <http:binding verb="GET"/>
  <wsdl:operation name="GetQuote">
    <http:operation location="/GetQuote"/>
    <wsdl:input>
      <http:urlEncoded/>
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

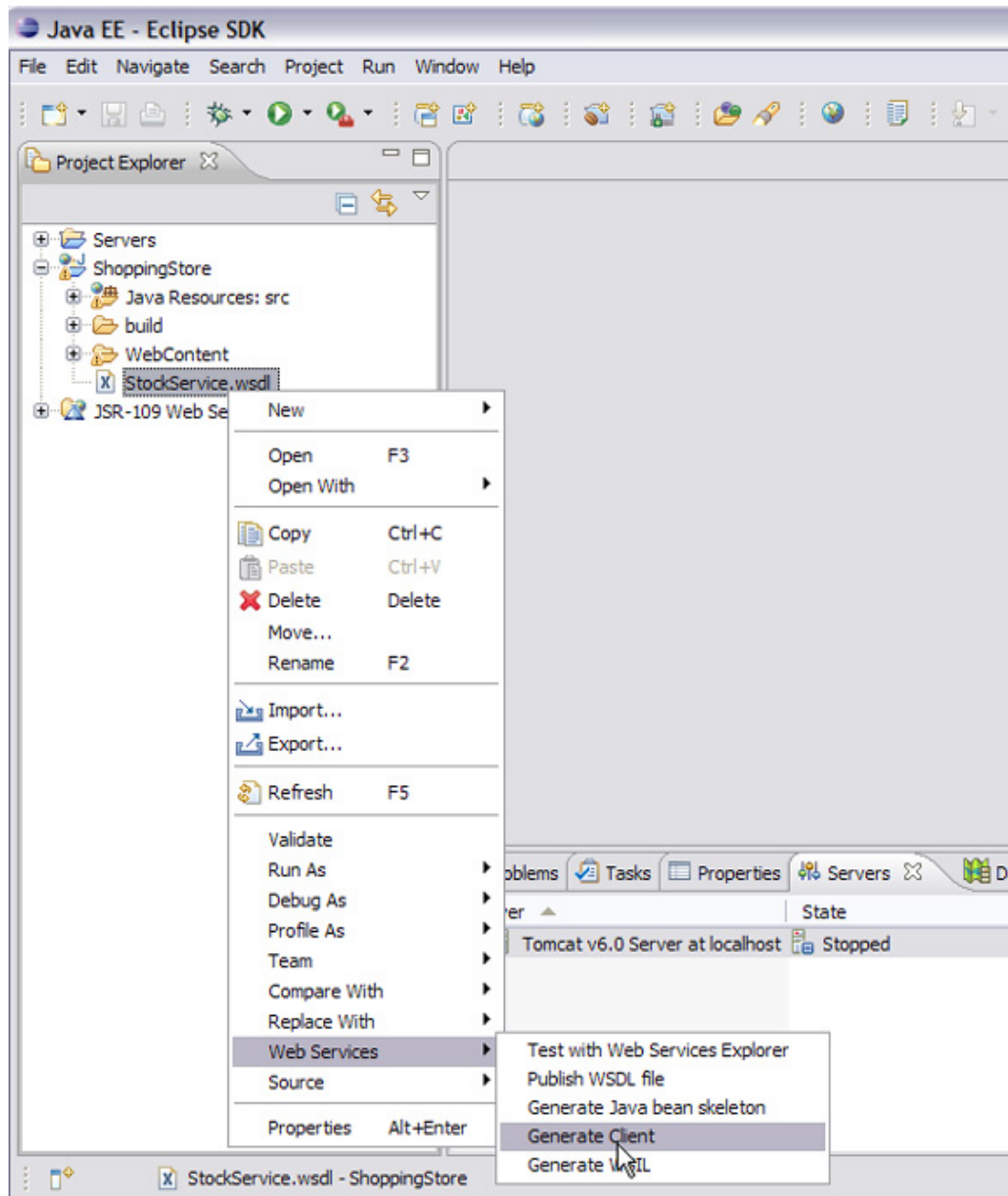
<wsdl:binding name="StockQuoteHttpPost" type="tns:StockQuoteHttpPost">
  <http:binding verb="POST"/>
  <wsdl:operation name="GetQuote">
    <http:operation location="/GetQuote"/>
    <wsdl:input>
      <mime:content part="NMTOKEN" type="application/x-www-form-urlencoded"/>
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="StockQuote">
  <wsdl:port name="StockQuoteSoap" binding="tns:StockQuoteSoap">
    <soap:address location="http://www.webservices.net/stockquote.asmx"/>
  </wsdl:port>
  <wsdl:port name="StockQuoteHttpGet" binding="tns:StockQuoteHttpGet">
    <http:address location="http://www.webservices.net/stockquote.asmx"/>
  </wsdl:port>
  <wsdl:port name="StockQuoteHttpPost" binding="tns:StockQuoteHttpPost">
    <http:address location="http://www.webservices.net/stockquote.asmx"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

2. Right-click on the **Stock Service.wsdl** file (shown in Figure 2), and select **Web Services -Generate Client**.

Figure 2. Generating Web service artifacts using StockService.wsdl

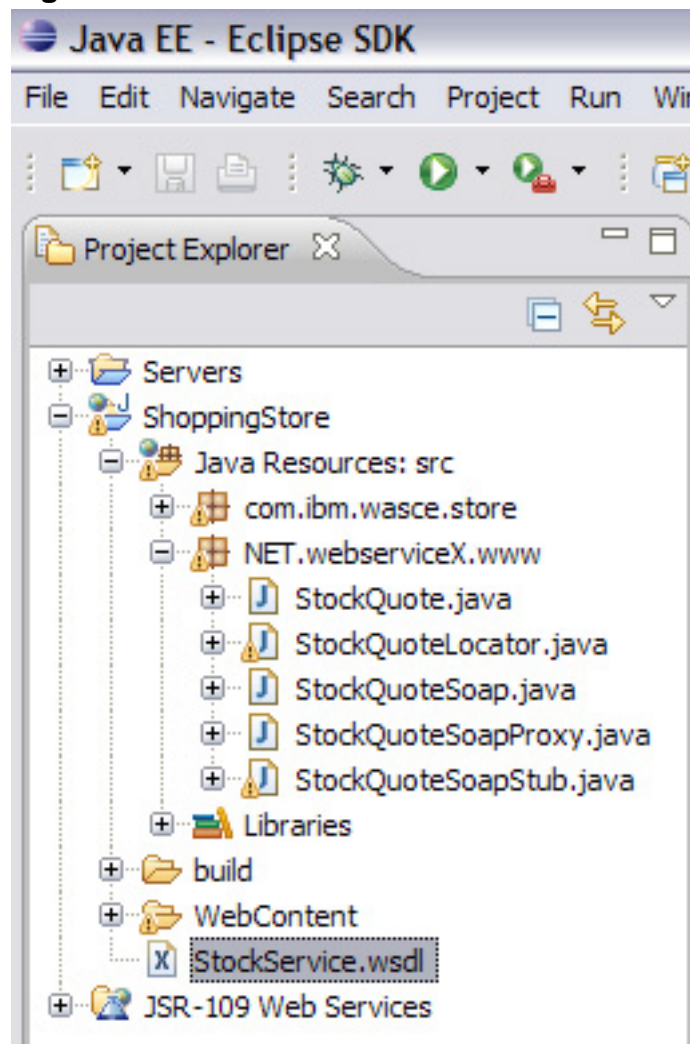


3. This step automatically generates all the required artifacts for the Web services. The following classes will be generated, as Figure 3 shows:
 - StockQuote.java
 - StockQuoteLocator.java

- StockQuoteSoap.java
- StockQuoteSoapProxy.java
- StockQuoteSoapStub.java

This step also downloads the JARs mentioned in section V ****add link**** and moves the jars to the `WEB-INF/lib` file of the application.

Figure 3. Generated Web services classes



4. To access the Web services in our application, we have the code from Listing 13 added to `checkoutcart.jsp` and `generalstore.jsp` files.
Listing 13. Code snippet to access the Stock Quote for IBM

```
<%  
StockQuoteLocator sql = new StockQuoteLocator();  
StockQuoteSoap sqs = sql.getStockQuoteSoap();
```

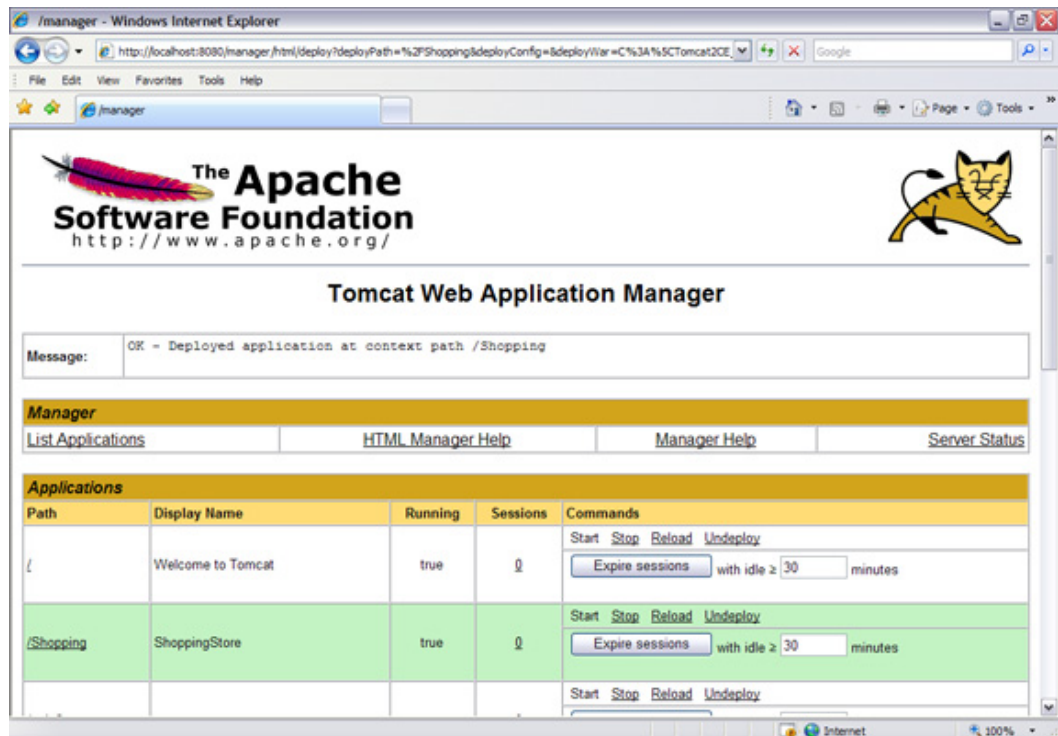
```
PrintWriter out1 =response.getWriter();
String s1=sqs.getQuote("IBM");
    s1=s1.substring(46,51);
%>
```

In our case we are passing `IBM` as the variable, which will retrieve the IBM stock quotes from the `Stock Web` service.

Deploying the application in Tomcat

In this section we will deploy the application in Tomcat, using these steps:

1. Export the application from Eclipse as a WAR file.
 2. Start Tomcat and Launch the Web Console using this command:
`http://localhost:8080/manager/html`
 3. You will be prompted to login; using the `UserDatabaseRealm` that we set up earlier. Since a manager role is needed, enter `tomcat` for the **user name** and **password**.
 4. The manager application shows all the deployed modules. Scroll to the bottom where it has a box for **Deploy directory or WAR file located on server**. For **Context path**, enter `/Shopping`, and for **WAR or Directory URL**, enter the absolute path to the `generalstore.war` file.
 5. Click the Deploy button. You should now see the General Store application running with the `/Shopping` context. Figure 4 shows the manager application after successful deployment
- Figure 4. Successful deployment of application in Tomcat**



- To access the application, enter this URL:
<http://localhost:8080/Shopping/store.cgi>.

Migrating the application to Community Edition

In general, to migrate the General Store application to Community Edition, you need to:

- Configure and replace the Tomcat `UserDatabaseRealm` to an authentication realm managed by the Community Edition container.
- Start and use the Derby RDBMS included in Community Edition, rather than the stand-alone version of Derby.
- Create a new data source managed by Community Edition.
- Create the required database table and fill it with data.
- Configure a JNDI reference to point to the Derby data source in Community Edition.
- Configure the ActiveMQ JMS broker connection factory and queue in Community Edition.

The elements that need not be changed during the migration are:

- JNDI reference to the data source in `web.xml`.
- Security configuration protecting the application in `web.xml`.

These elements are coded carefully to be portable between J2EE containers.

Changing ports used by Tomcat connectors in Community Edition

If you need to change the port used by Tomcat connectors in Community Edition, don't look for the `server.xml` file; there isn't one in Community Edition. Instead, take a look at the `config-substitutions.properties` file in the `var/config` directory under the Community Edition installation directory. Listing 14 shows the port configuration in Community Edition:

Listing 14. Tomcat port configuration in Community Edition

```
AJPPort=8009
clusterNodeName=NODE
ORBPort=6882
MaxThreadPoolSize=500
ResourceBindingsNamePattern=
SMTPHost=localhost
ResourceBindingsQuery=?\#org.apache.geronimo.naming.ResourceSource
COSNamingPort=1050
webcontainer=TomcatWebContainer
OpenEJBPort=4201
ORBSSLPort=2001
PortOffset=0
ActiveMQStompPort=61613
JMXPort=9999
ORBHost=localhost
EndPointURI=http://localhost:8080
NamingPort=1099
DefaultWadiSweepInterval=36000
WebConnectorConTimeout=20000
HTTPSPort=8443
COSNamingHost=localhost
MinThreadPoolSize=200
ReplicaCount=2
ServerHostname=0.0.0.0
ActiveMQPort=61616
ORBSSLHost=localhost
SMTPPort=25
webcontainerName=tomcat6
ResourceBindingsNameInNamespace=jca:
DefaultWadiNumPartitions=24
HTTPPort=8080
clusterName=CLUSTER_NAME
ClusterName=DEFAULT_CLUSTER
ResourceBindingsFormat={groupId}/{artifactId}/{j2eeType}/{name}
RemoteDeployHostname=localhost
TmId=71,84,77,73,68
```

You can modify the code from Listing 14 in Community Edition to change the default port configuration. Also you can uncomment the `PortOffset` variable and input a

value of your choice. This change modifies the Port settings for all the services in Community Edition by the defined value of `PortOffset`. For example, if you define `PortOffset=10`, then your HTTP port will be 8090 and the URL for the administrative console would be <http://localhost:8090/console>.

Remember to shutdown the standalone Derby database used by the Tomcat application, by using this command:

```
java org.apache.derby.drda.NetworkServerControl stop
```

Alternatively, you can use the Web console instead to add a Tomcat connector, or to edit the port number used with the connector. Just select **Server - Web Server** from the Web console menu. After editing and saving the port number change, use restart to allow the port change to take effect immediately. Remember that if you are using that port number for the administrative console display, change the port number in your browser URL to redisplay the console.

Configuring a Community Edition managed authentication realm

By default, Community Edition manages a realm called `geronimo-admin`, which uses the properties file under the `var/security` directory to store the user, password, and group information. The only user configured in this realm is `system` with a password of `manager`, belonging to the `admin` group and role `.` You can add users and groups to the default realm by using the **Security - Console Realm** option from the Web console menu.

You can configure this security realm in the Community Edition-specific deployment plan `geronimo-web.xml`. You need to add the code from Listing 15 in `geronimo-web.xml`:

Listing 15. Security realm configuration in Community Edition

```
<security-realm-name>geronimo-admin</security-realm-name>
<sec:security>
  <sec:default-principal realm-name="geronimo-properties-realm">
    <sec:principalclass="org.apache.geronimo.
      security.realm.providers.GeronimoUserPrincipal" name="system"/>
  </sec:default-principal>
  <sec:role-mappings>
    <sec:role role-name="admin">
      <sec:realm realm-name="geronimo-admin">
        <sec:principalclass="org.apache.geronimo.
          security.realm.providers.GeronimoGroupPrincipal" name="admin"/>
        <sec:principalclass="org.apache.geronimo.
          security.realm.providers.GeronimoUserPrincipal" name="system"/>
      </sec:realm>
    </sec:role>
  </sec:role-mappings></sec:security>
```

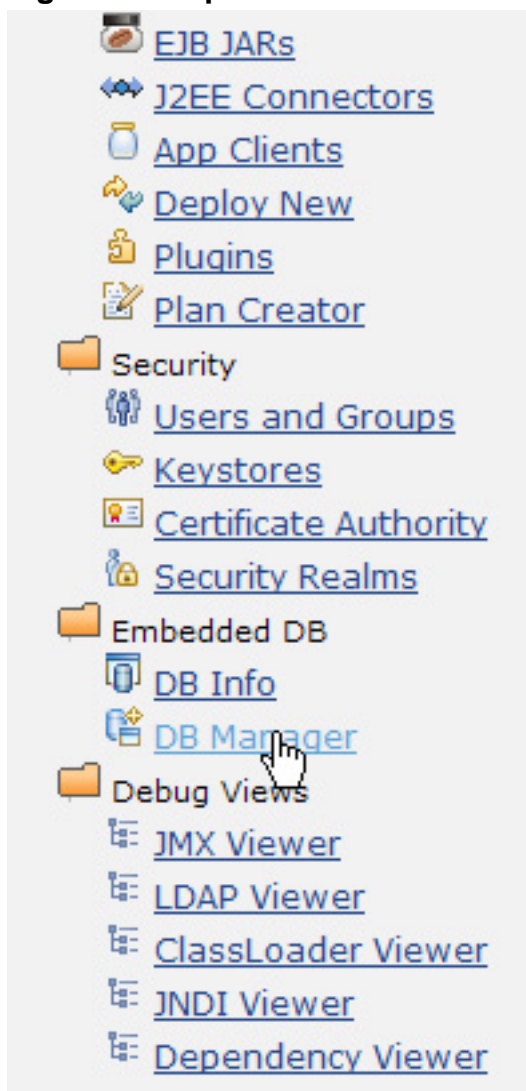
Creating the Promotion database table in Community Edition

You can easily create the Promotion table in Community Edition using a GUI-based

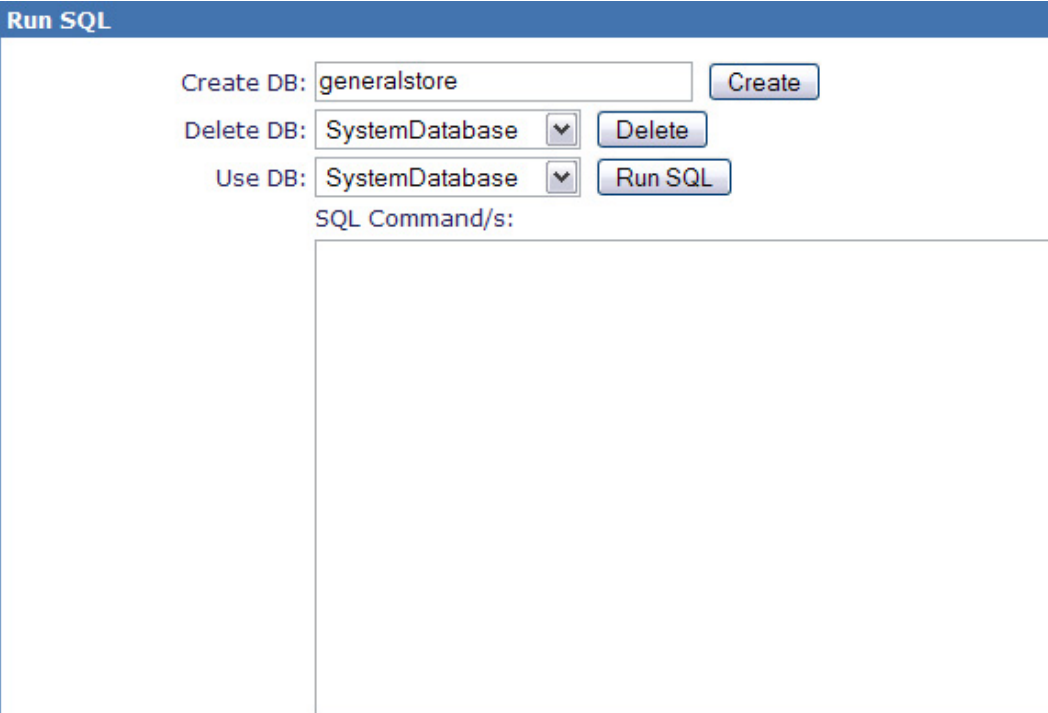
Web console:

1. Start the server.
2. Launch the administrative console using: `http://localhost:8080/console`.
3. Enter default **user name** (`system`) and **password** (`manager`).
4. In the console navigation, under **Embedded DB**, select **DB Manager** as Figure 5 shows:

Figure 5. DB portlet in administrative console



5. On the next screen (Figure 6), name the **Create DB** `generalstore` and select **Create**.

Figure 6. Creating the generalstore database

The screenshot shows a web interface titled "Run SQL". It contains three rows of controls:

- Create DB:** A text input field containing "generalstore" and a "Create" button.
- Delete DB:** A dropdown menu showing "SystemDatabase" and a "Delete" button.
- Use DB:** A dropdown menu showing "SystemDatabase" and a "Run SQL" button.

Below these controls is a large text area labeled "SQL Command/s:". At the bottom left of the interface, there is a "Note:" section with three numbered instructions:

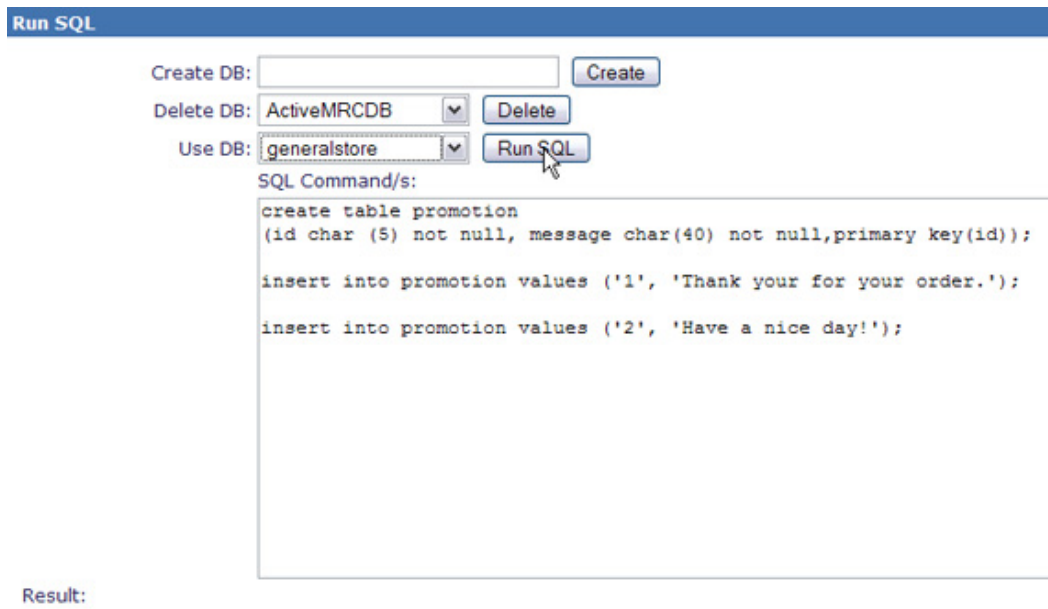
- 1) Use ';' to separate multiple statements
- 2) Query results will be displayed for single 'Select' statement
- 3) Use single quotes to encapsulate literal strings

6. From the **Use DB** drop down box, select **generalstore** and run the script show in Listing 16 and Figure 7:

Listing 16. Script to create promotion table in Community Edition

```
create table promotion
(id char (5) not null,
 message char(40) not null, primary key(id));
insert into promotion values ('1', 'Thank your for your order.');
```

Figure 7. Creating the promotion table



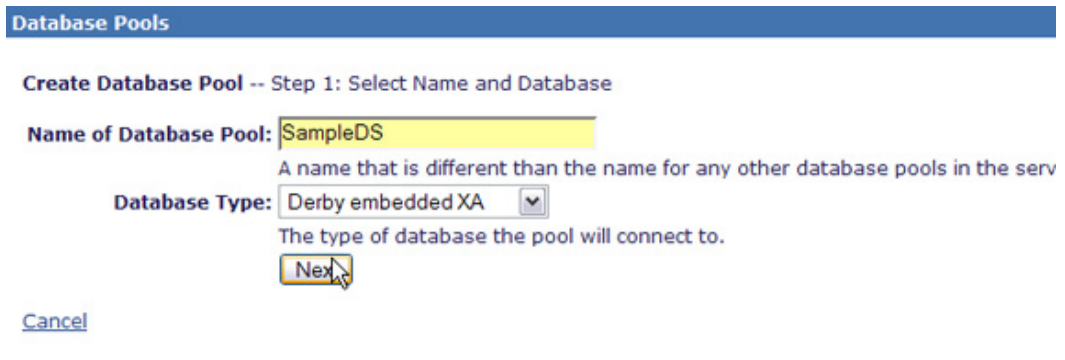
Creating a Derby data source in Community Edition

1. Launch the administrative console using <http://localhost:8080/console>. Enter the default **user name** and **password** (system and manager).
2. In the console navigation, under **Services** select **Database Pools**.
3. On the next screen (Figure 8) select **Using Geronimo database pool wizard**:

Figure 8. Selecting the Geronimo database pool wizard



4. Name the pool `SampleDS` and set the **Database Type** to **Derby Embedded XA**, as in Figure 9. Select **Next**.
Figure 9. Creating a Database pool in Community Edition



Database Pools

Create Database Pool -- Step 1: Select Name and Database

Name of Database Pool: A name that is different than the name for any other database pools in the serv

Database Type: The type of database the pool will connect to.

[Cancel](#)

5. On the Database Pools wizard, fill the form as Figure 10 shows. Note the here the **Password** is `APP`. Click **Deploy**.
Figure 10. Creating Database pool in Community Edition

Database Pools

This page lists all the available database pools.

For each pool listed, you can click the **usage** link to see examples of how to use the pool from your application.

Name	Deployed As
MonitoringClientDS	Server-wide
NoTxDataSource	Server-wide
SampleDS	Server-wide
SystemDataSource	Server-wide
jdbc/ActiveDS	Server-wide
jdbc/ArchiveDS	Server-wide
jdbc/juddiDB	org.apache.geronimo.configs/uddi-tomcat/2.1.1/car

Create a new database pool:

- ◆ [Using the Geronimo database pool wizard](#)
- ◆ [Import from JBoss 4](#)
- ◆ [Import from WebLogic 8.1](#)

7. You associate SampleDS to the web.xml JNDI reference of jdbc/storeDB using a <dependency> and <resource-ref> element in the geronimo-web.xml deployment plan, as shown in Listing 16.
- Listing 17. Code snippet from geronimo-web.xml deployment plan**

```
<sys:dependency>
  <sys:groupId>console.dbpool</sys:groupId>
  <sys:artifactId>SampleDS</sys:artifactId>
</sys:dependency>

<naming:resource-ref>
  <naming:ref-name>jdbc/storeDB</naming:ref-name>
  <naming:resource-link>SampleDS</naming:resource-link>
</naming:resource-ref>
```

Creating an ActiveMQ JMS Connection Factory and Queue

You can easily create a Community Edition-managed Connection Factory and Queue using the GUI Web console:

1. Launch the administrative console. In the console navigation, under **Services**, select **JMS Resources**. On the next screen select **For ActiveMQ**.
2. On the next screen fill in the form as shown in Figure 12. Click **Next**. **Figure 12. Configuring a JMS resource group**

JMS Resource Group -- Configure Server Connection

The settings on this screen are different for each JMS provider, but they generally configure connectivity to the JMS server. Cor settings to communicate with the server.

Resource Group Name: migration
A unique name for the resource adapter; used to generate the configuration name for th using the settings on this page.

Basic Configuration Settings.

ServerUrl: tcp://localhost:61616
The URL to the ActiveMQ server that you want this connection to connect to. If using an

UserName: defaultUser
The default user name that will be used to establish connections to the ActiveMQ server.

Password: defaultPassword
The default password that will be used to log the default user into the ActiveMQ server.

Clientid:
The client id that will be set on the connection that is established to the ActiveMQ server

UseInboundSession: false
Boolean to configure if outbound connections should reuse the inbound connection's ses

BrokerXmlConfig:
Sets the XML configuration file used to configure the embedded ActiveMQ broker via Sprin on the classpath unless a URL is specified. So a value of foo/bar.xml would be assumed t URL string is supported.

UseInboundSession: false
Boolean to configure if outbound connections should reuse the inbound connection's ses

DurableTopicPrefetch: 100
The maximum number of messages sent to a consumer on a durable topic until acknowle

QueuePrefetch: 1000
The maximum number of messages sent to a consumer on a queue until acknowledgmer

InputStreamPrefetch: 100
The maximum number of messages sent to a consumer on a JMS stream until acknowle

TopicPrefetch: 32766
The maximum number of messages sent to a consumer on a non-durable topic until ackn

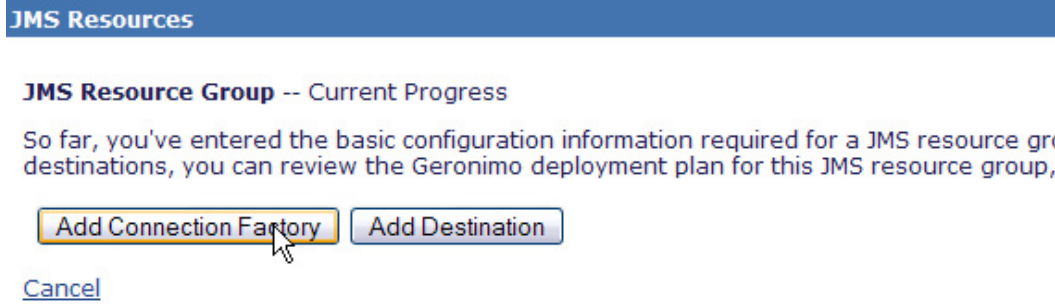
InitialRedeliveryDelay: 1000
The delay before redeliveries start. Default: 1000

MaximumRedeliveries: 5
The maximum number of redeliveries or -1 for no maximum. Default: 5

RedeliveryBackOffMultiplier: 5
The multiplier to use if exponential back off is enabled. Default: 5

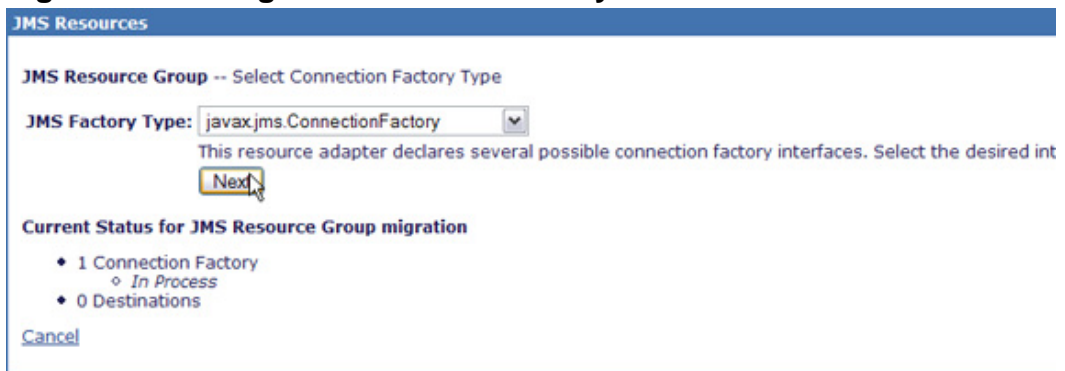
RedeliveryUseExponentialBackOff: false
To enable exponential backoff. Default: false

3. On the next screen (Figure 13) select **Add Connection Factory**
Figure 13. Adding a Connection Factor



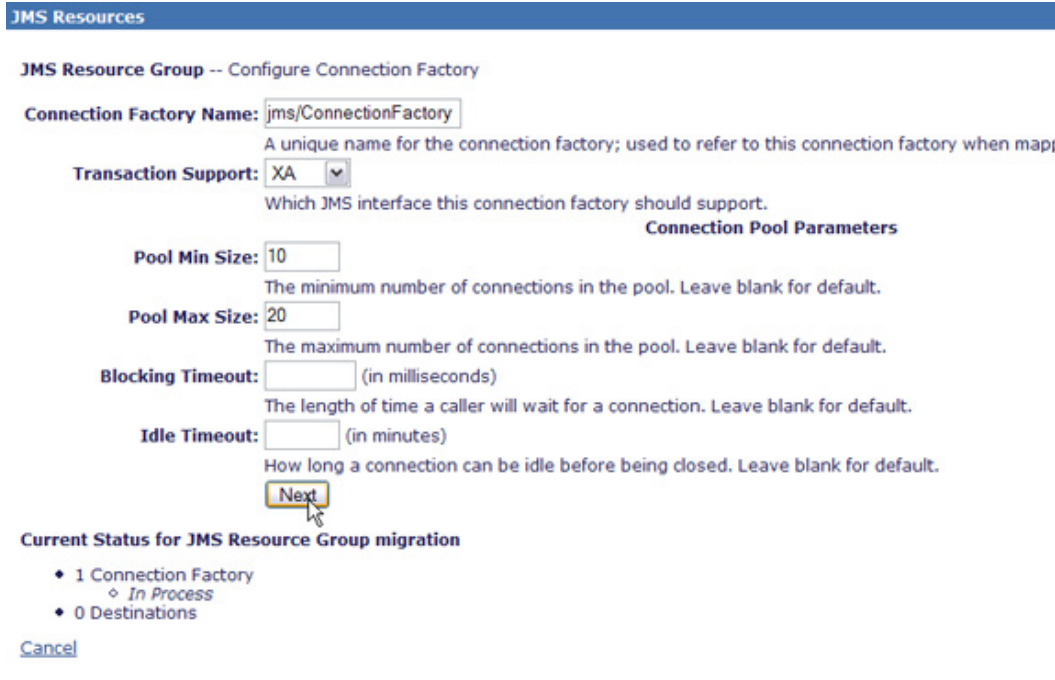
4. Select **javax.jms.ConnectionFactory** as the factory type (Figure 14) and click **Next**

Figure 14. Adding a Connection Factory



5. Fill in the form as shown in Figure 15. Click **Next**.

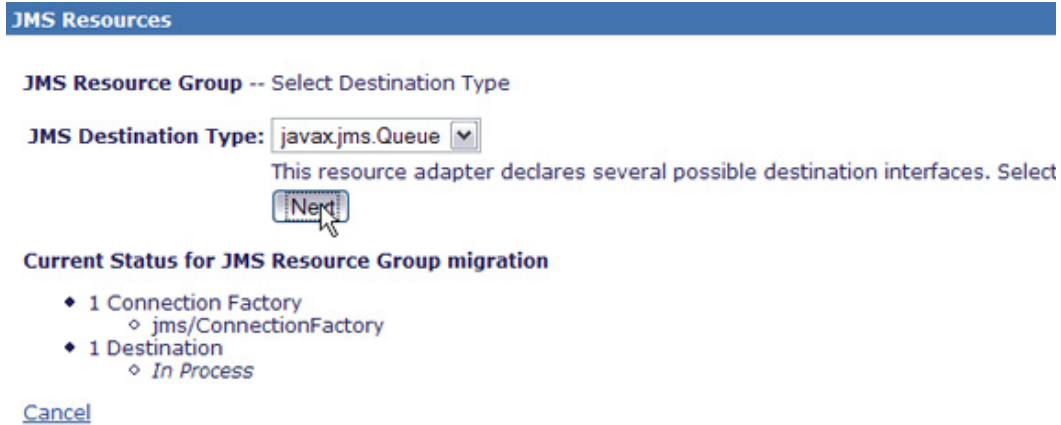
Figure 15. Adding a Connection Factory



- 6. On the next screen (Figure 16), select **Add Destination** to add a queue.
Figure 16. Adding a Destination

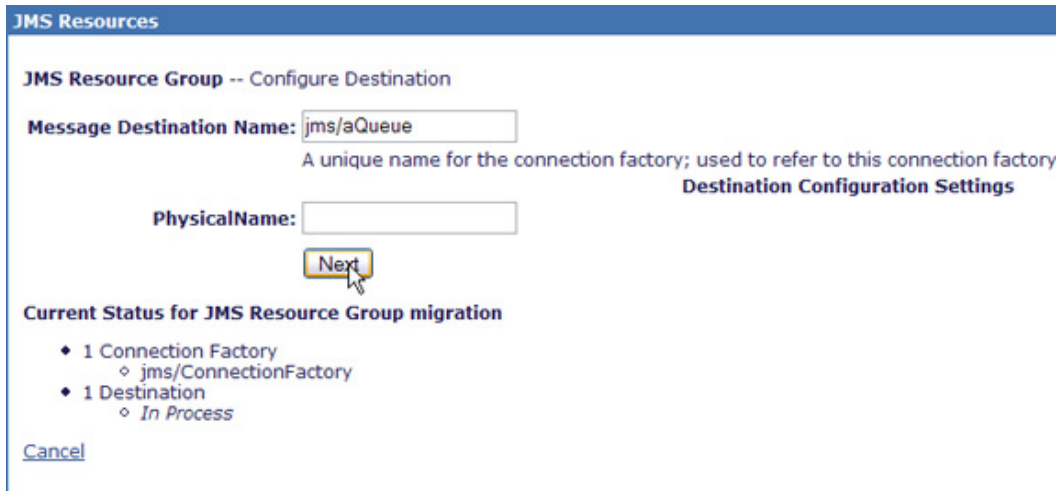


- 7. From the drop down box select **javax.jms.Queue** as Figure 17 shows. Click **Next**.
Figure 17. Setting the destination type



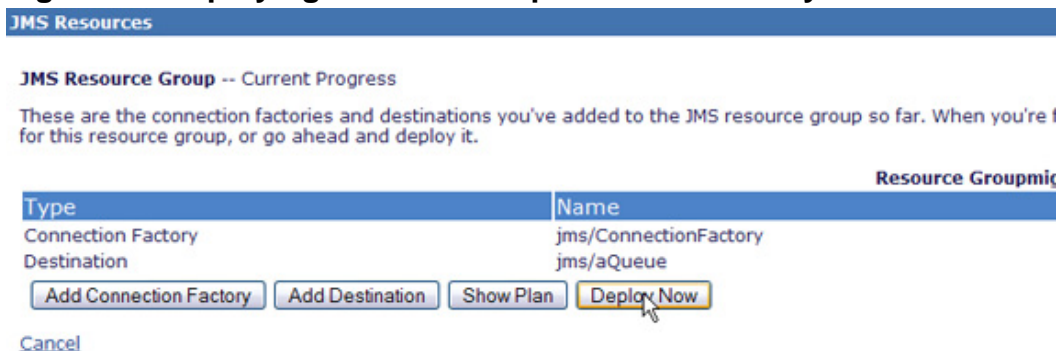
- Set the **Message Destination Name** to `jms/aQueue` as Figure 18 shows. Click **Next**.

Figure 18. Adding a Queue



- On the next screen (Figure 19) select **Deploy Now**, which deploys the resource plan on Community Edition.

Figure 19. Deploying the resource plan to Community Edition



- On successful deployment you can see the migration as one of the

available JMS resources, as Figure 20 shows:

Figure 20. Migration JMS resource in the list of available resources

JMS Resources	
This page lists all the available JMS Resource Groups.	
ActiveMQ RA (org.apache.geronimo.configs/activemq-ra/2.1.1/car)	
Type	Name
Connection Factory	DefaultActiveMQConnectionFactory
Queue	MDBTransferBeanOutQueue
Queue	SendReceiveQueue
migration (console.jms/migration/1.0/rar)	
Type	Name
Connection Factory	jms/ConnectionFactory
Queue	jms/aQueue
Create a new JMS Resource Group:	
<ul style="list-style-type: none"> ◆ For ActiveMQ ◆ For another JMS provider... 	

- You can associate the connection factory and queue that are already in web.xml by using <dependency> and <resource-ref> elements in the geronimo-web.xml deployment plan, as shown in Listing 18.

Listing 18. Code snippet from geronimo-web.xml deployment plan

```
<sys:dependency>
  <sys:groupId>org.apache.geronimo.configs</sys:groupId>
  <sys:artifactId>activemq-ra</sys:artifactId>
  <sys:version>2.1.1</sys:version>
  <sys:type>car</sys:type>
</sys:dependency>
  ...
  ...
<naming:resource-ref>
  <naming:ref-name>jms/ConnectionFactory</naming:ref-name>
  <naming:pattern>
    <naming:groupId>org.apache.geronimo.configs</naming:groupId>
    <naming:artifactId>activemq-ra</naming:artifactId>
    <naming:version>2.1.1</naming:version>
    <naming:name>DefaultActiveMQConnectionFactory</naming:name>
  </naming:pattern>
</naming:resource-ref>

<naming:message-destination>
  <naming:message-destination-name>
    aQueue
  </naming:message-destination-name>
  <naming:admin-object-link>
    SendReceiveQueue
  </naming:admin-object-link>
</naming:message-destination>
```

Removing unneeded services from the Community Edition

WAR file

If you compare the size of the sample `ShoppingStore_Tomcat.war` file (approximately 9MB) with the size of the `ShoppingStore_CommunityEdition.war` file (approximately 32KB) you may wonder why there is such a huge size difference. The reason is that all the services (Axis, JMS, commons, etc.) and Derby database drivers that you need to include in the Tomcat installed application `WEB-INF\lib` directory to are already integrated into Community Edition. The advantage of removing these jar files is that you do not have to integrate and support them.

The following jar files from `ShoppingStore_Tomcat.war` are included in `WEB-INF\lib`:

- `activeio-core-3.0.0-incubator.jar`
- `apache-activemq-4.1.1.jar`
- `axis.jar`
- `commons-discovery-0.2.jar`
- `commons-logging.jar`
- `derby.jar`
- `derbyclient.jar`
- `derbynet.jar`
- `derbytools.jar`
- `jaxrpc.jar`
- `jstl.jar`
- `saaj.jar`
- `standard.jar`
- `wsdl4j-1.5.1.jar`

These jar files are also included in the Community Edition `<WASCE_HOME>/repository` directory. To determine if the JAR exists, search in the `/repository` directory for the component name without the version (e.g. `activeio-core`), or without the source (e.g. `activemq`). The corresponding jar file in the `/repository` will contain the version (e.g. `axis1-4.jar`). JAR files that are not listed in the `/repository` (for example application-specific files) would need to be maintained in `WEB-INF\lib`.

In this migration example, all of the JAR files in the Tomcat `WEB-INF\lib` directory can be removed. Most Web applications place some application specific JAR files in the `WEB-INF\lib` directory, so the ability to reduce the library will vary. Database driver JAR files should generally be installed in the `\repository` and shared with applications. The Administrative console tool will help download and install database driver files (i.e. Oracle or MySQL) that are not available in the `\repository` (select **Services - Database Pools - Using the Geronimo database pool wizard.**)

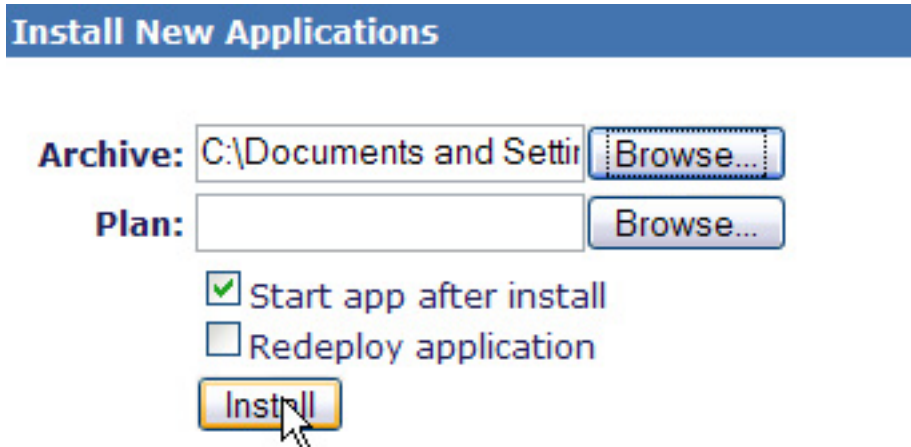
Also, there may be `<servlet>` parameters within the `web.xml` file that point to the JAR files in the `WEB-INF\lib` directory. You need to determine if removing a JAR file will require changes to `web.xml`.

Deploying the application in Community Edition

To deploy the application, launch the administrative console and follow these steps:

1. In the console navigation under **Applications**, select **Deploy New**.
2. Select **Browse** beside the **Archive** text box and browse to `Shoppingstore.war`. Click **Install**. This step deploys the application on Community Edition, as Figure 21 shows:

Figure 21. Deploying an application on Community Edition



3. Launch the application using <http://localhost:8080/Shopping/store.cgi>. When prompted for login, use `system` for the **user name** and `manager` for the **password**.

Troubleshooting migration problems

The sample application in this article provides examples of configuration and deployment plans for a typical Tomcat migration. Many different resource types are included in the sample as deployment examples. The **Deploy New** menu in the administrative console provides a detailed error message for a WAR file that cannot deploy.

The scripted deploy tool also provides an error if the WAR file cannot be deployed. If errors occur during application testing, review the Eclipse Server log or the `var\log\server.log` file (`geronimo.out` file for Linux) for Java errors that may occur when accessing the application.

Some common problems with migration are:

- Classloader issues
- Resource configuration issues
- Deployment plan issues

We'll look at these issues in detail in the following sections.

Classloader issues

By default Community Edition follows a classloader hierarchy that loads classes from the parent first (i.e. from the `\repository`). Tomcat loads classes from the application first. Issues may occur if the Tomcat application uses older components (ex. an older version of Axis) and the API structure has changed. In this case, when you deploy the WAR or at runtime, classloader errors may occur.

One method to correct classloader issues is to add `hidden-classes` parameters to the deployment plan after the `<moduleId>` parameter. The `hidden-classes` identify class names that should not be loaded from the parent configurations. Listing 19 shows a `geronimo-web.xml` example for an Axis classloader problem. The `axis.jar` file in `WEB-INF\lib` would be used for the application:

Listing 19. The hidden-classes code for an Axis classloader problem

```
</moduleId>
<hidden-classes>
  <filter>org.apache.axis2</filter>
</hidden-classes>
```

Another method to correct classloader issues is to add the `inverse-classloading` parameter to the deployment plan after the `<moduleId>` parameter. Inverse classloading causes Community Edition to follow a classloader hierarchy that loads classes from the parent immediately after the application `WEB-INF\lib` classes. Listing 20 shows a `geronimo-web.xml` example is

included below for inverse-classloader:

Listing 20. The inverse-classloading code for a Geronimo classloader problem

```
</moduleId>  
<inverse-classloading/>
```

Modifying the classloader structure should allow the application to migrate successfully. However, you should correct the application code at some point to work with components in Community Edition, since integration and support of components are key advantages for using Community Edition over Tomcat. For more information, visit [Managing the classpath](#) from the documentation.

Resource configuration issues

Resources like Database and Security connections can cause deploy and runtime issues. Since Tomcat has a different configuration structure, using `server.xml` and `context.xml`, configuration issues may occur when moving to Community Edition. In most cases, the issue is an incorrect parameter or access problem. For databases, try using the **Database Pools - Geronimo database pool wizard** in the administrative console to test the database connection. When you **Deploy** the database pool, it tests against the database and an error occurs if the connection is unsuccessful. You can edit the pool to change parameters and retest until the connection is successful. For security realms, try using the **Security Realms - Add new security realm** menu in the administrative console to test the security realm connection. A menu in the Security Realm wizard lets you enter a log-in username and password to test the connection. If the log in is unsuccessful, you can edit the security realm and retest until it is successful.

Deployment Plan issues

The `geronimo-web.xml` deployment plan requires specific parameters and format to work correctly. The General Store application used in this article has a valid `geronimo-web-xml` file included in the `ShoppinStore_WASCE.war` file. A Sample Applications package is available at the [Community Edition download site](#). These samples include various application types with a `geronimo-web.xml` deployment plan. You can use these samples as a basis for your own deployment plan requirements.

Also, try using the **Plan Creator** menu in the administrative console to create the `geronimo-web.xml` deployment plan from the Tomcat WAR file and resource connections (i.e. Database, JMS, Security) created by the administrative console. This option provides a simple method for creating a `geronimo-web.xml` deployment plan. When deploying a WAR file, you can define the deployment plan outside of the WAR file by using the **Plan** field. See [Figure 21](#) for information on

deploying the WAR file and external deployment plan.

Conclusion

Migrating applications from Tomcat 6.0.x to Community Edition V2.1 is straightforward, because Tomcat 6.0.x is integrated, intact, and actually part of Community Edition. However, since Community Edition now controls all configuration management, configuration elements typically found in `server.xml` or `context.xml` on Tomcat are now migrated to:

- **config-substitution.properties**: for quick change parameters, such as port number for connectors.
- **geronimo-web.xml**: the Community Edition-specific deployment plan for an application context; can be included in the `WEB-INF` directory of the archive.

Stepping through the migration of a Web application from Tomcat 6.x to WebSphere Application Server Community Edition, we have seen that the major points of this migration include the switching of authentication realms, JNDI reference mapping, set up of a Derby database, configuration of JDBC connectors, JMS resource factory and JMS queue.

Downloads

Description	Name	Size	Download method
Tomcat Web archive	ShoppingStore_Tomcat.war	8.9MB	HTTP
Web archive migrated to Community Edition	ShoppingStore_CommunityEdition.war	8.9MB	HTTP

[Information about download methods](#)

Resources

Learn

- [Best Practices Tomcat to Geronimo Migration](#)
- [What's new in WebSphere Application Server Community Edition V2.1](#)
- [Developing JPA Applications with WebSphere Application Server Community Edition](#)
- [Community Edition support site](#)
- [WebSphere Application Server Community Edition documentation](#)
- [WebSphere Application Server Community Edition samples](#)
- [Apache Geronimo site](#)
- [Develop applications on all the Java EE5 APIs](#)
- [Get to know Java EE 5](#)
- [Clustering in Community Edition](#)
- [Apache Geronimo samples](#)
- [WebSphere Application Server Community Edition Technical Support offerings](#)
- [WebSphere Application Server Community Edition resources](#)
- [developerWorks Open Source zone](#)

Get products and technologies

- [Download Community Edition](#)
- [Community Edition Eclipse plug-in update](#)
- [ActiveMQ](#)
- [Apache Tomcat](#)
- [OpenEJB](#)
- [Apache Axis2](#)

Discuss

- [developerWorks WebSphere Application Server Community Edition and Apache Geronimo forum](#)
- [developerWorks Open Source forum](#)

About the author

Ashish Jain



Ashish Jain is a Software Engineer working for Level 3 Technical Support of IBM WebSphere Application Server Community Edition. He received a Bachelors of Engineering in Computer Science from NITK Surathkal, and joined IBM in 2005 as an ELTP.