Plan Creator

{scrollbar}

To facilitate the creation of Geronimo-specific deployment plans there is a new portlet now available. The **Plan Creator** wizard available from the Geronimo Administration Console makes deployment easier by walking you through a sequence of steps to auto-generate the geronimo-web.xml for a given **WAR** file.

The wizard searches the references declared in the `web.xml` as well as what is now defined within the code itself via annotations. Depending on the type of application the Plan Creator wizard will present a series of pages requesting specific information to resolve the references and dependencies. The wizard also offers default values which work for most of the configurations.

Depending on the type of application you are deploying, you may be presented with six separate pages to address the application's specific configuration. These pages are (in sequence):

1. Load WAR
2. WAR - Environment
3. WAR - References
4. WAR - Security
5. WAR - Dependencies
6. Created Plan and Deploy WAR

This document offers a series of different sample application so you can have a better understanding of how this wizard works. This document is organized in the following sections:

Supported features

The **Plan Creator** wizard will help you generate the appropriate deployment plan for the application you are deploying. The wizard currently works for web apps and supports the following features.

- **References**: EJB, EJB Local, JDBC Connection Pool, JMS Connection Factory, JMS Destination, JavaMail Session & Web Service references declared in the web-apps are auto discovered and users are asked to resolve them by listing Available Resources in the server environment to which they can be linked.
- Above type of references declared inside the Java classes through **Annotations** are also auto discovered.
- Simplified configuration of **Security**.

The following sections provide sample applications with different features so you can better appreciate the wizard behavior.

Sample with annotations annotations

There are four simple applications attached to give you a better idea of how the plan creator wizard would work when deploying web applications.

- JDBC access
- EJB access
- JMS access

JDBC access jdbc

This sample is a very simple database access application that uses annotations. For this example we have created a sample **BankDB** database on the embedded Derby as well as a database connection pool. Although we will not be covering in this section how to create a database or a connection pool we still provide in the #Attachments section some SQL sample to generate the required sample database.

The Annotations-TestJDBCAccess.zip file provides a `BankDB.sql`, **`WebAppJDBCAccessAnnotations.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `WebAppJDBCAccessAnnotations_generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

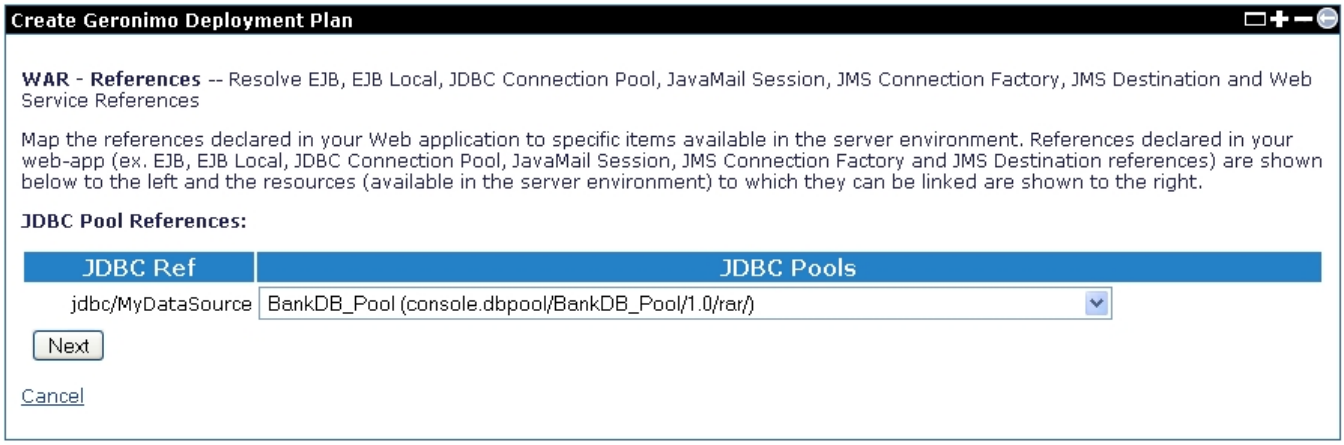
`WebAppJDBCAccessAnnotations.war` includes a Servlet, a JSP and a basic web.xml. The following excerpt shows the annotation part the wizard has identified and will likely be prompting for additional information while deploying the application.

javasolidExcerpt from ListCustomers.java ... import javax.annotation.Resource; ... /** * Servlet implementation class for Servlet: ListCustomers * */ public class ListCustomers extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet { @Resource(name = "jdbc/MyDataSource") private DataSource dataSource; ...

With the server up and running access the Geronimo Administration Console with a web browser and click on **Plan Creator** on the left menu.

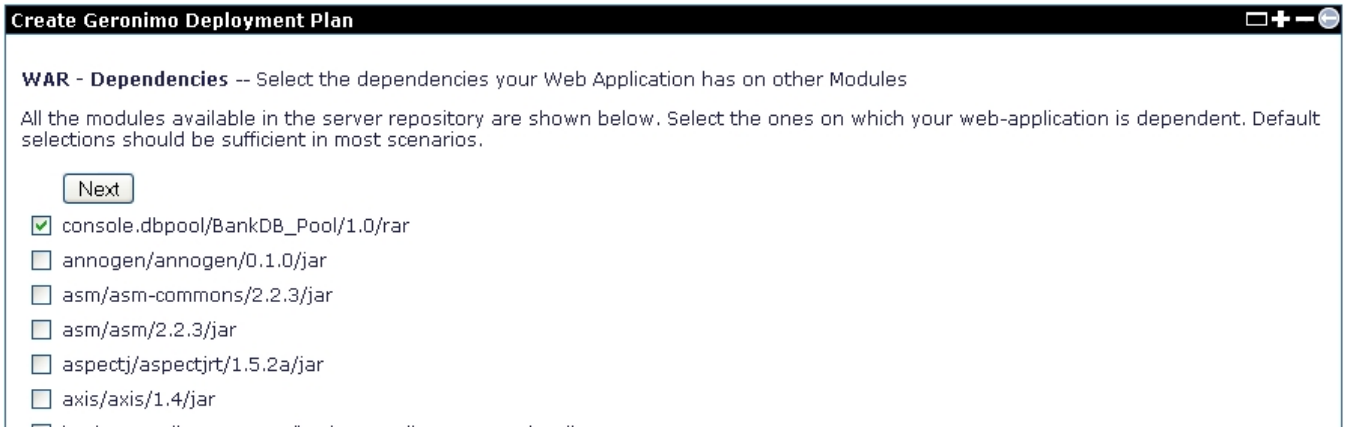
Browse to the `WebAppJDBCAccessAnnotations.war` you just extracted and click on **Configure**. In the resulting screen you will be configuring the web application identity. You may want to change the default proposed values however, for this example, we will be accepting the defaults. Click **Next**.

In the following screen, the wizard would have identified all WAR references that need to be resolved. The references listed on this screen are specific to this application. The **JDBCRef** column is showing the resource name defined in `ListCustomers.java`. See **`@Resource(name = "jdbc/MyDataSource")`** from the excerpt above.



As mentioned earlier, for this example we created a database connection pool we called **BankDB_Pool**. From the **JDBC Pools** pull-down menu you can now select the appropriate connection pool and click **Next**.

The following screen list all available modules and allows you specify the modules this WAR has dependencies on. This screen also provides a default selection, in this case the database connection pool will be already selected. Accept the default and click **Next**. For most scenarios the default values should be sufficient.



The final configuration page displays the generated deployment plan and allows you to make any additional editing. The following example shows the generated deployment plan.

xmlsolidGenerated deployment plan <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep: artifactId>WebAppJDBCAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep: dependencies> <dep:dependency> <dep:groupId>console.dbpool</dep:groupId> <dep:artifactId>BankDB_Pool</dep:artifactId> <dep:version>1.0</dep: version> <dep:type>rar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJDBCAccessAnnotations< /context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jdbc/MyDataSource</nam:ref-name> <nam:pattern> <nam:groupId>console.dbpool</nam:groupId> <nam:artifactId>BankDB_Pool</nam:artifactId> <nam:version>1.0</nam:version> <nam: name>BankDB_Pool</nam:name> </nam:pattern> </nam:resource-ref> </web-app>

The last step is to actually deploy and start the application, click on **Deploy WAR**.

You should receive two confirmation messages stating the application was successfully deployed and successfully started. The **Launch Web App** link takes your browser directly to the application you just deployed based on the context root you defined earlier. **Finish** takes you to the Plan Creator portlet again to start deploying a new application.

EJB access ejb

This sample is a simple web application that accesses a Session EJB. It requires a JAR to be deployed before you can deploy the web application. In the # Attachments section we have included all the files you need to test the deployment of this sample application.

The Annotations-TestEJBAccess.zip file provides the `CurrencyConverterEJB.jar` that needs to be deployed first, the **`WebAppEjbAccessAnnotations.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `WebAppEjbAccessAnnotations_generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

With the server up and running access the Geronimo Administration Console with a web browser and click on **Deploy New** on the left menu. We will now deploy the WAR prerequisite JAR file.

On the **Archive:** field browse to the `CurrencyConverterEJB.jar` file you just extracted and click **Install**, this will install and start the JAR. You can verify the status of this JAR by clicking on **EJB JARs** link from the menu on the left. It should display something like this:

Installed EJB JARs

Expert User (enable all actions on Geronimo Provided Components)

Component Name	State	Commands			Parent Components
default/CurrencyConverterEJB/1199912901187/jar	running	Stop	Restart	Uninstall	org.apache.geronimo.configs/axis/2.1-SNAPSHO org.apache.geronimo.configs/axis2/2.1-SNAPSH org.apache.geronimo.configs/j2ee-corba-yoko/2 org.apache.geronimo.configs/openejb/2.1-SNAP org.apache.geronimo.configs/openjpa/2.1-SNAP org.apache.geronimo.configs/system-database/ org.apache.geronimo.configs/tomcat6/2.1-SNAP
org.apache.geronimo.configs/mejb/2.1-SNAPSHOT/car	running	Stop	Restart	Uninstall	org.apache.geronimo.configs/openejb/2.1-SNAP org.apache.geronimo.configs/system-database/

`WebAppEjbAccessAnnotations.war` includes a Servlet, a JSP and a basic web.xml. The following excerpt shows the annotation part the wizard has identified and will likely be prompting for additional information while deploying the application.

javasolidExcerpt from ConverterHandler.java ... public class ConverterHandler extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet { @EJB(name = "ejb/Converter") private Converter converter; ...

With the prerequisite JAR just installed click on **Plan Creator** on the left menu to proceed with the WAR installation. Browse to the `WebAppEjbAccessAnnotations.war` you just extracted and click on **Configure**. In the resulting screen you will be configuring the web application identity. You may want to change the default proposed values however, for this example, we will be accepting the defaults. Click **Next**.

In the following screen, the wizard would have identified all WAR references that need to be resolved. The references listed on this screen are specific to this application. The **EJBRef** column is showing the resource name defined in `ConverterHandler.java`. See **`@EJB(name = "ejb/Converter")`** from the excerpt above.

From the **EJBs Deployed** pull-down menu select the JAR you deployed earlier **`ConverterBean (default/CurrencyConverterEJB/1199912901187/jar)`** and click **Next**.

Create Geronimo Deployment Plan

WAR - References -- Resolve EJB, EJB Local, JDBC Connection Pool, JavaMail Session, JMS Connection Factory, JMS Destination and Web Service References

Map the references declared in your Web application to specific items available in the server environment. References declared in your web-app (ex. EJB, EJB Local, JDBC Connection Pool, JavaMail Session, JMS Connection Factory and JMS Destination references) are shown below to the left and the resources (available in the server environment) to which they can be linked are shown to the right.

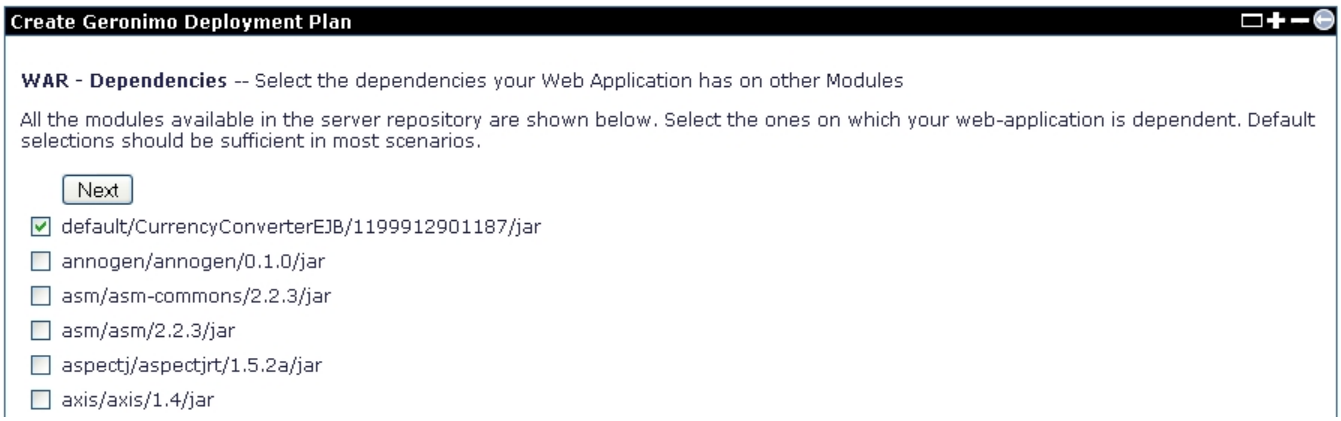
EJB References:

EJB Ref	EJBs Deployed
ejb/Converter	ConverterBean (default/CurrencyConverterEJB/1199912901187/jar/)

Next

Cancel

The following screen list all available modules and allows you specify additional modules this WAR has dependencies on. This screen also provides a default selection, in this case the EJB JAR you deployed previously will be already selected. Accept the default and click **Next**. For most scenarios the default values should be sufficient.



The final configuration page displays the generated deployment plan and allows you to make any additional editing. The following example shows the generated deployment plan.

xmlsolidGenerated deployment plan <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep: artifactId>WebAppEjbAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep: dependencies> <dep:dependency> <dep:groupId>default</dep:groupId> <dep:artifactId>CurrencyConverterEJB</dep:artifactId> <dep: version>1199912901187</dep:version> <dep:type>jar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppEjbAccessAnnotations</context-root> <nam:ejb-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>ejb /Converter</nam:ref-name> <nam:pattern> <nam:groupId>default</nam:groupId> <nam:artifactId>CurrencyConverterEJB</nam:artifactId> <nam: version>1199912901187</nam:version> <nam:name>ConverterBean</nam:name> </nam:pattern> </nam:ejb-ref> </web-app>

The last step is to actually deploy and start the application, click on **Deploy WAR**.

You should receive two confirmation messages stating the application was successfully deployed and successfully started. The **Launch Web App** link takes your browser directly to the application you just deployed based on the context root you defined earlier. **Finish** takes you to the Plan Creator portlet again to start deploying a new application.

JMS access jms

This is a simple web application that accesses a JMS Connection Factory and a JMS destination. **Producer** servlet sends 5 messages to a JMS queue and the **Consumer** servlet synchronously receives messages from the same queue.

The Annotations-TestJMSAccess.zip file provides the **`WebAppJMSAccessAnnotations.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `WebAppJMSAccessAnnotations_generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

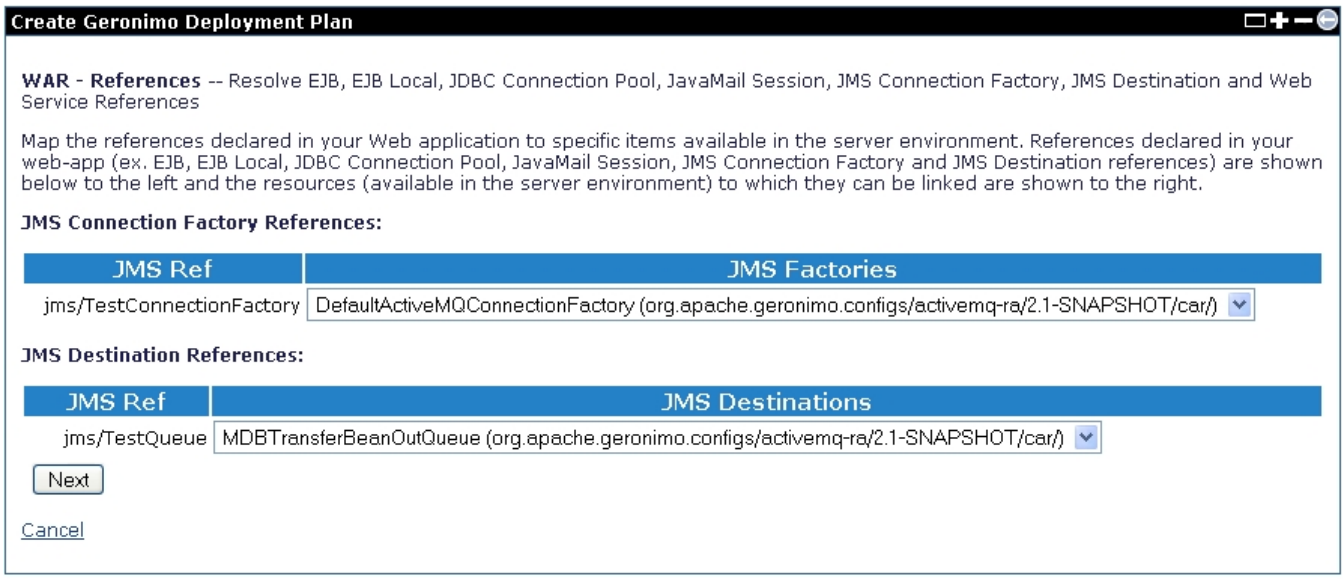
`WebAppJMSAccessAnnotations.war` includes the two Servlets mentioned above, a JSP and a basic web.xml. The following excerpts show the annotation part the wizard has identified and will likely be prompting for additional information while deploying the application.

javasolidExcerpt from ProducerServlet.java ... /** * Servlet implementation class for Servlet: ProducerServlet * */ public class ProducerServlet extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet { @Resource(name="jms/TestConnectionFactory") private ConnectionFactory connectionFactory; @Resource(name="jms/TestQueue") private Queue queue; ... javasolidExcerpt from ConsumerServlet.java ... /** * Servlet implementation class for Servlet: ConsumerServlet * */ public class ConsumerServlet extends javax.servlet.http.HttpServlet implements javax.servlet. Servlet { @Resource(name="jms/TestConnectionFactory") private ConnectionFactory connectionFactory; @Resource(name="jms/TestQueue") private Queue queue; ...

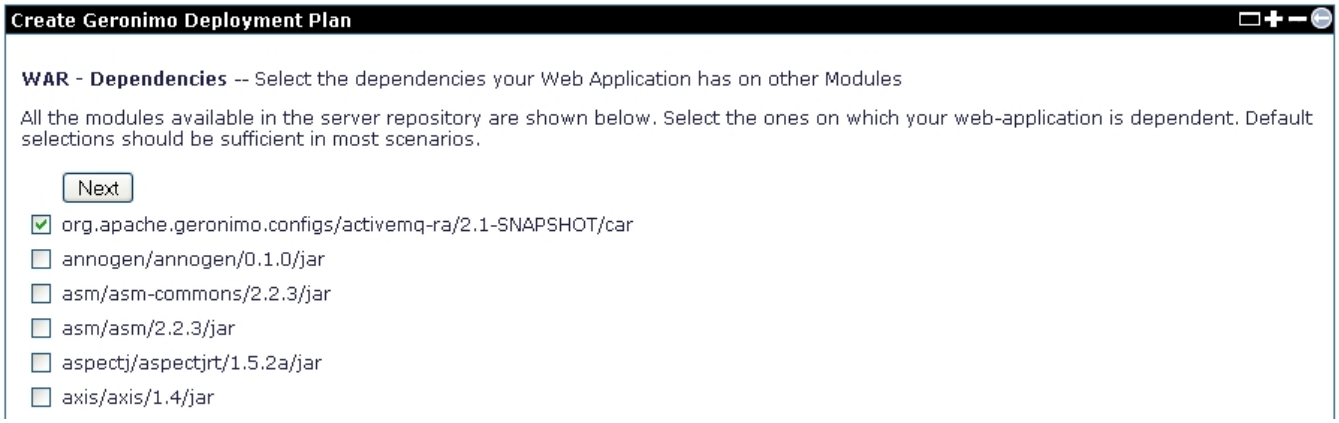
With the server up and running access the Geronimo Administration Console with a web browser and click on **Plan Creator** on the left menu.

Browse to the `WebAppJMSAccessAnnotations.war` you just extracted and click on **Configure**. In the resulting screen you will be configuring the web application identity. You may want to change the default proposed values however, for this example, we will be accepting the defaults. Click **Next**.

In the following screen, the wizard would have identified all WAR references that need to be resolved. The references listed on this screen are specific to this application. The **JMS Ref** column is showing the resource names defined in the servlets shown above. See **`@Resource(name="jms /TestConnectionFactory")`** and **`@Resource(name="jms/TestQueue")`**. The following image shows the factories and destinations already defined in Geronimo, you may want to define your own factories and destinations for your application specifically before running this wizard. For this example we will just accept the defaults and click **Next**.



The following screen list all available modules and allows you specify the modules this WAR has dependencies on. This screen also provides a default selection, in this case the database connection pool will be already selected. Accept the default and click **Next**. For most scenarios the default values should be sufficient.



The final configuration page displays the generated deployment plan and allows you to make any additional editing. The following example shows the generated deployment plan.

xmlsolidGenerated deployment plan <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppJMSAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-ra</dep:artifactId> <dep:version>2.1-SNAPSHOT</dep:version> <dep:type>car</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJMSAccessAnnotations</context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jms/TestConnectionFactory</nam:ref-name> <nam:pattern> <nam:groupId>org.apache.geronimo.configs</nam:groupId> <nam:artifactId>activemq-ra</nam:artifactId> <nam:version>2.1-SNAPSHOT</nam:version> <nam:name>DefaultActiveMQConnectionFactory</nam:name> </nam:pattern> </nam:resource-ref> <nam:resource-env-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jms/TestQueue</nam:ref-name> <nam:pattern> <nam:groupId>org.apache.geronimo.configs</nam:groupId> <nam:artifactId>activemq-ra</nam:artifactId> <nam:version>2.1-SNAPSHOT</nam:version> <nam:name>MDBTransferBeanOutQueue</nam:name> </nam:pattern> </nam:resource-env-ref> </web-app>

The last step is to actually deploy and start the application, click on **Deploy WAR**.

You should receive two confirmation messages stating the application was successfully deployed and successfully started. The **Launch Web App** link takes your browser directly to the application you just deployed based on the context root you defined earlier. **Finish** takes you to the Plan Creator portlet again to start deploying a new application.

So far we have shown you how easy is to deploy a web application when using annotations. The following section focuses on applications not using annotations, this is mainly for applications based on the previous J2EE specification.

Sample without annotations noannotations

To make it easier to compare we are providing a similar set of applications as in the previous section(JDBC, EJB and JMS) but without the advantage of using annotations. In addition we are also including a security configuration application to cover the additional functionality we didn't cover in the previous section.

JDBC access jdbc2

This is basically the same sample application we used in the Sample with annotations section, but obviously without annotations. Once again for this example we created a sample database and a connection pool. However, the emphasis of this section is on the **web.xml** file which contains the resource reference required for this application to run.

The noAnnotations-TestJDBCAccess.zip file provides a `BankDB.sql`, **`WebAppJDBCAccess.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

`WebAppJDBCAccess.war` includes a Servlet, a JSP and `web.xml`. The following sample illustrates the `web.xml`, in this example the Plan Creator wizard will look into the **`<resource-ref>`** section.

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"> <servlet> <description></description> <display-name>ListCustomers</display-name> <servlet-name>ListCustomers</servlet-name> <servlet-class>myPackage.ListCustomers</servlet-class> </servlet> <servlet-mapping> <servlet-name>ListCustomers</servlet-name> <url-pattern>/listCustomers</url-pattern> </servlet-mapping> <resource-ref> <res-ref-name>jdbc/MyDataSource</res-ref-name> <res-type>javax.sql.DataSource</res-type> <res-auth>Container</res-auth> <res-sharing-scope>Shareable</res-sharing-scope> </resource-ref> </web-app>

With the server up and running access the Geronimo Administration Console with a web browser and click on **Plan Creator** on the left menu. Browse to the `WebAppJDBCAccess.war` you just extracted and click on **Configure**. The subsequent configuration pages are identical to what we described in the Sample with annotations section so we will skip the step-by-steps details. The end result is the same, the Plan Creator wizard has generated the deployment plan directly from the provided **web.xml**. To facilitate comparison we are providing both generated deployment plans side-by-side, note that only the **`<artifactId>`** and **`<context-root>`** are different in order to make these applications unique when deployed on the same server.

50% xmlsolidGenerated deployment plan without Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppJDBCAccess</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>console.dbpool</dep:groupId> <dep:artifactId>BankDB_Pool</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>rar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJDBCAccess</context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jdbc/MyDataSource</nam:ref-name> <nam:pattern> <nam:groupId>console.dbpool</nam:groupId> <nam:artifactId>BankDB_Pool</nam:artifactId> <nam:version>1.0</nam:version> <nam:name>BankDB_Pool</nam:name> </nam:pattern> </nam:resource-ref> </web-app> 50% xmlsolidGenerated deployment plan with Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppJDBCAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>console.dbpool</dep:groupId> <dep:artifactId>BankDB_Pool</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>rar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJDBCAccessAnnotations</context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jdbc/MyDataSource</nam:ref-name> <nam:pattern> <nam:groupId>console.dbpool</nam:groupId> <nam:artifactId>BankDB_Pool</nam:artifactId> <nam:version>1.0</nam:version> <nam:name>BankDB_Pool</nam:name> </nam:pattern> </nam:resource-ref> </web-app>

EJB access ejb2

This example is based on the same sample application we used in the Sample with annotations section but without annotations. In the #Attachments section we have included all the files you need to test the deployment of this sample application.

The noAnnotations-TestEJBAccess.zip file provides the `CurrencyConverterEJB.jar` that needs to be deployed first, the **`WebAppEjbAccess.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

`WebAppEjbAccess.war` includes a Servlet and the `web.xml`. In the annotations example we provided a servlet as well but for this sample all the processing is done directly by the JSP. The following sample illustrates the `web.xml`, in this example the Plan Creator wizard will look into the **`<ejb-ref>`** section.

xmlsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app id="WebApp_ID" version="2.5" 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"> <ejb-ref> <ejb-ref-name>ejb/Converter</ejb-ref-name> <ejb-ref-type>Session</ejb-ref-type> <remote>myPackage.Converter</remote> </ejb-ref> </web-app>

`WebAppEjbAccess.war` requires you deploy the EJB JAR first. If you have not done some while testing the annotation samples make sure you deploy it now. With the server up and running access the Geronimo Administration Console with a web browser and click on **Deploy New** on the left menu. On the **Archive:** field browse to the `CurrencyConverterEJB.jar` file you just extracted and click **Install**, this will install and start the JAR.

After you installed the required EJB JAR click on **Plan Creator** on the left menu. Browse to the `WebAppEjbAccess.war` you just extracted and click on **Configure**. The subsequent configuration pages are identical to what we described in the Sample with annotations section so we will skip the step-by-steps details. The end result is the same, the Plan Creator wizard has generated the deployment plan directly from the provided **web.xml**. To facilitate comparison we are providing both generated deployment plans side-by-side, note that only the **`<artifactId>`** and **`<context-root>`** are different in order to make these applications unique when deployed on the same server.

50% xmlsolidGenerated deployment plan without Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppEjbAccess</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>default</dep:groupId> <dep:artifactId>CurrencyConverterEJB</dep:artifactId> <dep:version>1199912901187</dep:version> <dep:type>jar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppEjbAccess</context-root> <nam:ejb-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>ejb/Converter</nam:ref-name> <nam:pattern> <nam:groupId>default</nam:groupId> <nam:artifactId>CurrencyConverterEJB</nam:artifactId> <nam:version>1199912901187</nam:version> <nam:name>ConverterBean</nam:name> </nam:pattern> </nam:ejb-ref> </web-app> 50% xmlsolidGenerated deployment plan with Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppEjbAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>default</dep:groupId> <dep:artifactId>CurrencyConverterEJB</dep:artifactId> <dep:version>1199912901187</dep:version> <dep:type>jar</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppEjbAccessAnnotations</context-root> <nam:ejb-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>ejb/Converter</nam:ref-name> <nam:pattern> <nam:groupId>default</nam:groupId> <nam:artifactId>CurrencyConverterEJB</nam:artifactId> <nam:version>1199912901187</nam:version> <nam:name>ConverterBean</nam:name> </nam:pattern> </nam:ejb-ref> </web-app>

JMS access jms2

Once again we are basing this sample application from the one we used in the annotations section. There is a **Producer** servlet that sends 5 messages to a JMS queue and a **Consumer** servlet that synchronously receives messages from the same queue.

The noAnnotations-TestJMSAccess.zip file provides the **`WebAppJMSAccess.war`** which is the WAR we will be deploying and a sample of the generated deployment plan `generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

`WebAppJMSAccess.war` includes the two Servlets mentioned above, a JSP and the `web.xml`. The following sample illustrates the `web.xml`, in this example the Plan Creator wizard will look into the **`<resource-ref>`**,**`<message-destination-ref>`** and **`<message-destination>`** sections.

xmlsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"> <display-name> WebAppJMSAccess</display-name> <servlet> <description> </description> <display-name> ProducerServlet</display-name> <servlet-name>ProducerServlet</servlet-name> <servlet-class> com.apache.geronimo.samples.webAppJMSacess.ProducerServlet</servlet-class> </servlet> <servlet> <description> </description> <display-name> ConsumerServlet</display-name> <servlet-name>ConsumerServlet</servlet-name> <servlet-class> com.apache.geronimo.samples.webAppJMSacess.ConsumerServlet</servlet-class> </servlet> <servlet-mapping> <servlet-name>ProducerServlet</servlet-name> <url-pattern>/producer</url-pattern> </servlet-mapping> <servlet-mapping> <servlet-name>ConsumerServlet</servlet-name> <url-pattern>/consumer</url-pattern> </servlet-mapping> <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> <resource-ref> <res-ref-name>jms/TestConnectionFactory</res-ref-name> <res-type>javax.jms.ConnectionFactory</res-type> <res-auth>Container</res-auth> <res-sharing-scope>Shareable</res-sharing-scope> </resource-ref> <message-destination-ref> <message-destination-ref-name>jms/TestQueue</message-destination-ref-name> <message-destination-type>javax.jms.Queue</message-destination-type> <message-destination-usage>ConsumesProduces</message-destination-usage> <message-destination-link>TestQueue</message-destination-link> </message-destination-ref> <message-destination> <message-destination-name>TestQueue</message-destination-name> </message-destination> </web-app>

With the server up and running access the Geronimo Administration Console with a web browser and click on **Plan Creator** on the left menu. Browse to the `WebAppJMSAccess.war` you just extracted and click on **Configure**. The subsequent configuration pages are nearly identical to what we described in the Sample with annotations section so we will skip the step-by-steps details. The end result is the same, the Plan Creator wizard has generated the deployment plan directly from the provided **web.xml**.

Once again we are providing both generated deployment plans side-by-side to facilitate comparison. These two sample applications are slightly different so you can see some additional variations in the generated deployment plan.

50% xmlsolidGenerated deployment plan without Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppJMSAccess</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-ra</dep:artifactId> <dep:version>2.1-SNAPSHOT</dep:version> <dep:type>car</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJMSAccess</context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jms/TestConnectionFactory</nam:ref-name> <nam:pattern> <nam:groupId>org.apache.geronimo.configs</nam:groupId> <nam:artifactId>activemq-ra</nam:artifactId> <nam:version>2.1-SNAPSHOT</nam:version> <nam:name>DefaultActiveMQConnectionFactory</nam:name> </nam:pattern> </nam:resource-ref> <nam:message-destination xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:message-destination-name>TestQueue</nam:message-destination-name> <nam:admin-object-link>MDBTransferBeanOutQueue</nam:admin-object-link> </nam:message-destination> </web-app> 50% xmlsolidGenerated deployment plan with Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>WebAppJMSAccessAnnotations</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> <dep:dependencies> <dep:dependency> <dep:groupId>org.apache.geronimo.configs</dep:groupId> <dep:artifactId>activemq-ra</dep:artifactId> <dep:version>2.1-SNAPSHOT</dep:version> <dep:type>car</dep:type> </dep:dependency> </dep:dependencies> </dep:environment> <context-root>WebAppJMSAccessAnnotations</context-root> <nam:resource-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jms/TestConnectionFactory</nam:ref-name> <nam:pattern> <nam:groupId>org.apache.geronimo.configs</nam:groupId> <nam:artifactId>activemq-ra</nam:artifactId> <nam:version>2.1-SNAPSHOT</nam:version> <nam:name>DefaultActiveMQConnectionFactory</nam:name> </nam:pattern> </nam:resource-ref> <nam:resource-env-ref xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"> <nam:ref-name>jms/TestQueue</nam:ref-name> <nam:pattern> <nam:groupId>org.apache.geronimo.configs</nam:groupId> <nam:artifactId>activemq-ra</nam:artifactId> <nam:version>2.1-SNAPSHOT</nam:version> <nam:name>MDBTransferBeanOutQueue</nam:name> </nam:pattern> </nam:resource-env-ref> </web-app>

Security configuration security2

This sample application is a basic time reporting system that uses Servlets, JSPs and J2EE declarative security. In addition to above features it uses Geronimo's embedded Derby database to store user information of the system. Even though this application uses a database to hold user information, it is merely for configuration purposes. This sample application is a slight variation from the one covered in Geronimo v2.0 Documentation. The focus of this document is on the Plan Creator wizard so we will not cover in much detail this application's inner working, please refer to Web application security sample for further details.

The noAnnotations-TestSecuritySettings.zip provides a number of SQL scripts and deployment plans to facilitate set up the environment the security sample application requires to run. The provided files are `1_TimeReportDB.sql`, `2_dbPoolPlan.xml`, `3_securityRealmPlan.xml`. In addition there is a `0_Readme.txt` which is a condensed set of intructions to deploy this sample application, **`timereport.war`** which is the application we will be deploying and a sample of the generated deployment plan `timereport_generatedPlan.xml`. Download and extract the zip file to a directory of your convenience.

Before we continue with the security application itself we need to create some additional configurations using the provided plans.

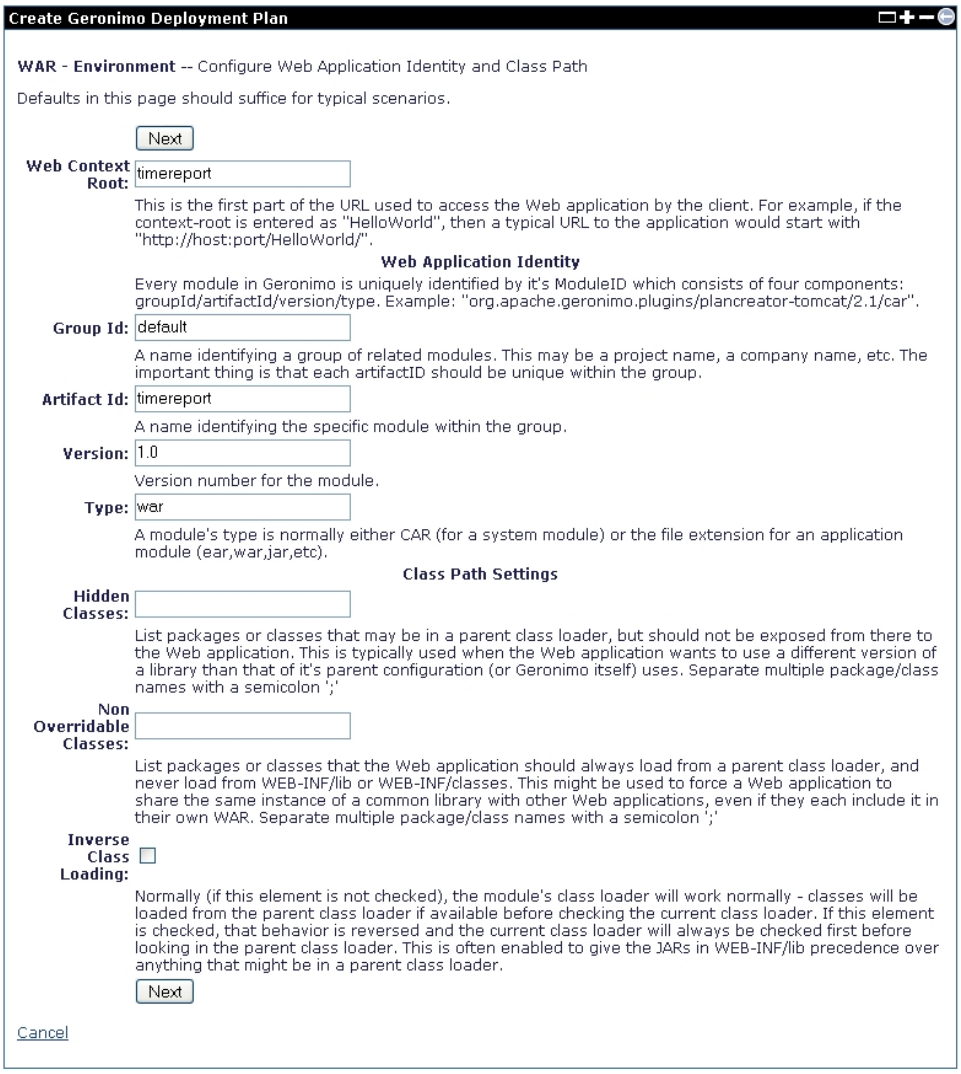
1. With the server up and running access the Geronimo Administration Console and click on **DB Manager**. Create a database by name "TimeReportDB" and run **`1_TimeReportDB.sql`** on this new database.
2. Create a database connection pool by clicking on **Deploy New** and specify **`<geronimo_home>\repository\org\tranql\tranql-connector-ra\1.3\tranql-connector-ra-1.3.rar`** as the **Archive:** and **`2_dbPoolPlan.xml`** as the **Plan**. This will create the new database pool "TimeReportPool".
3. Create a new security realm by clickin on **Deploy New** and only specify **`3_securityRealmPlan.xml`** as the "Plan". This will create the **TimeReportRealm** security realm.

Now that we have configured the environment we go back to the application. **`timereport.war`** provides 2 Servlets, several JSPs and the `web.xml` we will be focusing on. In addition this WAR contains the sources and full java docs. The following sample illustrates the `web.xml`, here are defined all the security constraints, realms and roles the Plan Creator wizard will use to generate the Geronimo specific deployment plan.

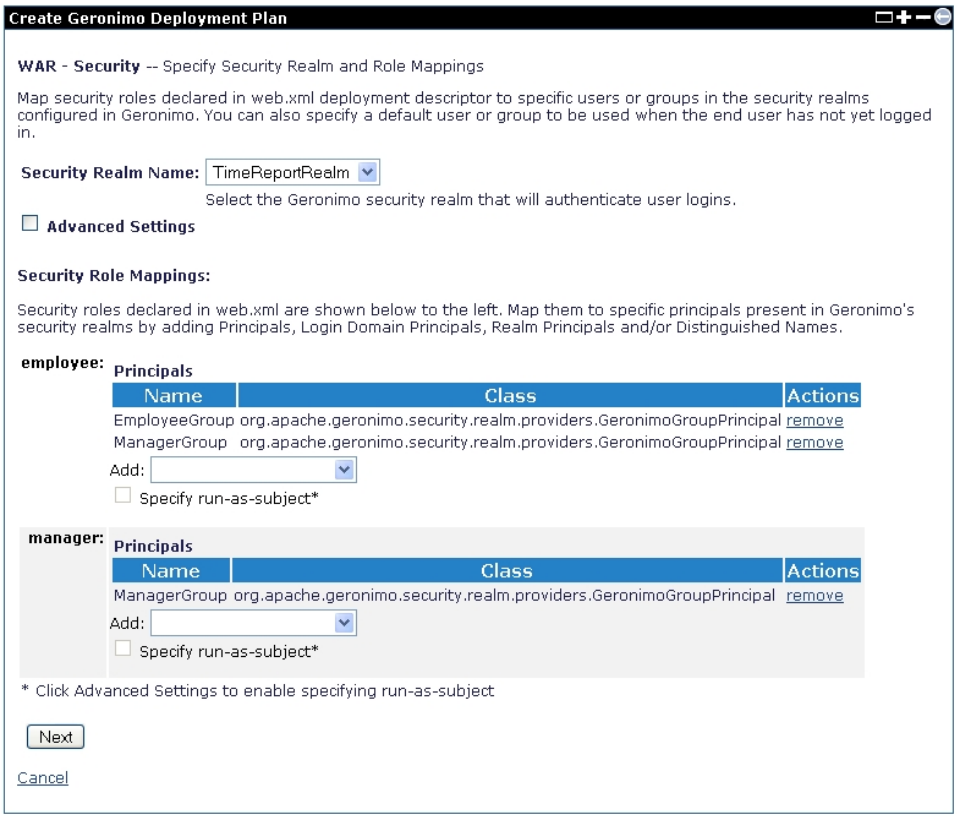
xmlsolidweb.xml <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001 /XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4"> <welcome-file-list> <welcome-file>index.html</welcome-file> </welcome-file-list> <security-constraint> <web-resource-collection> <web-resource-name>employee< /web-resource-name> <url-pattern>/employee/*</url-pattern> </web-resource-collection> <auth-constraint> <role-name>employee</role-name> </auth-constraint> </security-constraint> <security-constraint> <web-resource-collection> <web-resource-name>manager</web-resource-name> <url-pattern> /manager/*</url-pattern> </web-resource-collection> <auth-constraint> <role-name>manager</role-name> </auth-constraint> </security-constraint> <login-config> <auth-method>FORM</auth-method> <realm-name>TimeReportRealm</realm-name> <form-login-config> <form-login-page>/login/login.jsp< /form-login-page> <form-error-page>/login/login_error.jsp</form-error-page> </form-login-config> </login-config> <security-role> <role-name>employee< /role-name> </security-role> <security-role> <role-name>manager</role-name> </security-role> <servlet> <display-name>AddTimeRecordServlet< /display-name> <servlet-name>AddTimeRecordServlet</servlet-name> <servlet-class>org.apache.geronimo.samples.timereport.web. AddTimeRecordServlet</servlet-class> </servlet> <servlet> <display-name>AddEmployeeServlet</display-name> <servlet-name>AddEmployeeServlet< /servlet-name> <servlet-class>org.apache.geronimo.samples.timereport.web.AddEmployeeServlet</servlet-class> </servlet> <servlet-mapping> <servlet-name>AddTimeRecordServlet</servlet-name> <url-pattern>/employee/add_timerecord</url-pattern> </servlet-mapping> <servlet-mapping> <servlet-name>AddEmployeeServlet</servlet-name> <url-pattern>/manager/add_employee</url-pattern> </servlet-mapping> </web-app>

1. Back in the Geronimo Administration Console click on **Plan Creator**. Browse to the `timereport.war` you just extracted and click on **Configure**.

2. In the resulting screen you will be configuring the web application identity. You may want to change the default proposed values however, for this example, we will be accepting the defaults. Click **Next**.



3. As a difference from the other sample applications, for this example we do not have to resolve any resource references so the wizard will jump directly to the security part of the configuration. In the resulting screendefine the security configuration.
    a. Select "TimeReportRealm" from the "Security Realm Name:" pull down menu. This is the security realm you deployed earlier.
    b. For security role **employee** select **Add** -> **Principal** from the pull down menu and add 2 **Group Prinicipals** with name **EmployeeGroup** and **ManagerGroup**.
    c. For security role **manager**, select **Add** -> **Principal** and add 1 **Group Prinicipal** with name **ManagerGroup**.



4. Click **Next**, the following screen allows you to select WAR dependencies. This application has not dependencies on any these modules, click **Next**.
5. In the **Created Plan** screen you should see a plan similar to this one. xmlsolidGenerated deployment plan without Annotations <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"> <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"> <dep:moduleId> <dep:groupId>default</dep:groupId> <dep:artifactId>timereport</dep:artifactId> <dep:version>1.0</dep:version> <dep:type>war</dep:type> </dep:moduleId> </dep:environment> <context-root>timereport</context-root> <security-realm-name>TimeReportRealm</security-realm-name> <app:security xsi:type="sec:securityType" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:app="http://geronimo.apache.org/xml/ns/j2ee/application-2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <sec:role-mappings> <sec:role role-name="employee"> <sec:principal name="EmployeeGroup" class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"/> <sec:principal name="ManagerGroup" class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"/> </sec:role> <sec:role role-name="manager"> <sec:principal name="ManagerGroup" class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"/> </sec:role> </sec:role-mappings> </app:security> </web-app>
6. Click on **Deploy WAR** and then click on **Launch Web App** to run the sample application. Verify using **userid** and **password** values from **1_Time ReportDB.sql**.

Attachments

.*zip