BEA WebLogic Server™

MedRec Development Tutorial

Release 8.1
Document Date: February 2003
Revised: June 28, 2006

Copyright

Restricted Rights Legend

Trademarks or Service Marks

MedRec Tutorials

Part Number	Date	Software Version
N/A	March 28, 2003	BEA WebLogic Server Version 8.1

Configuring Domains and Servers

Building the MedRec Applications

Moving to Production Mode

Overview of the Avitek Medical Records Development Tutorials

The Avitek Medical Records Development Tutorials guide you through the process of developing, packaging, and deploying real-world J2EE applications with WebLogic Server. These tutorials use the Avitek Medical Records sample application suite (Version 1.1.1) as a basis for instruction. However, you can easily apply the procedures and best practices to your own J2EE applications.

What Is Avitek Medical Records?

Avitek Medical Records (or MedRec) is a WebLogic Server sample application suite that concisely demonstrates all aspects of the J2EE platform. MedRec is designed as an educational tool for all levels of J2EE developers; it showcases the use of each J2EE component, and illustrates best practice design patterns for component interaction and client development.

The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients. Patient data includes:

- Patient profile information—A patient's name, address, social security number, and login information.
- Patient medical records—Details about a patient's visit with a physician, such as the patient's vital signs and symptoms as well as the physician's diagnosis and prescriptions.

The MedRec application suite consists of two main J2EE applications and one supporting application that loads the MedRec informational page. The main applications support one or more user scenarios for MedRec:

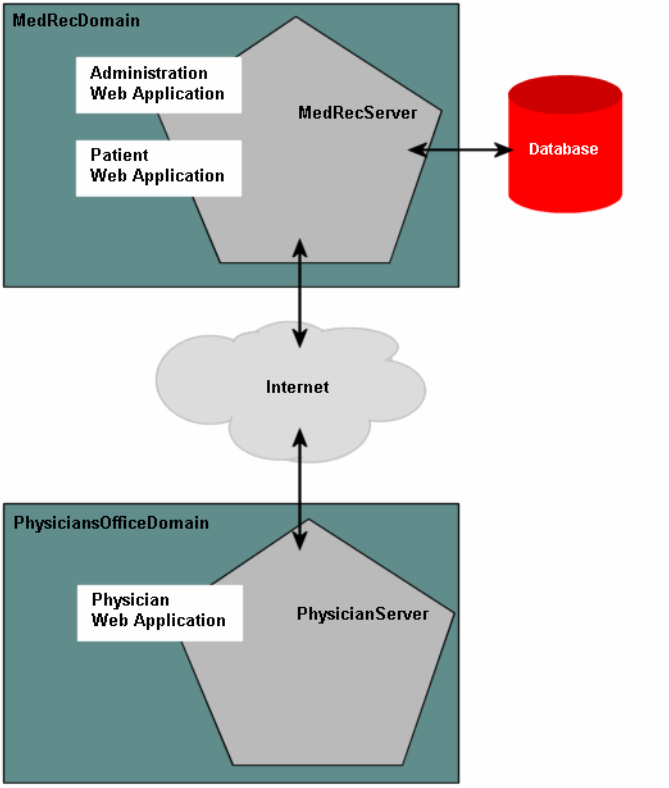
- `medrecEar`—Patients log in to the patient Web Application (`patientWebApp`) to edit their profile information, or request that their profile be added to the system. Patients can also view prior medical records of visits with their physician. Administrators use the administration Web Application (`adminWebApp`) to approve or deny new patient profile requests.

`medrecEar` also provides all of the controller and business logic used by the MedRec application suite, as well as the Web Service used by different clients.

- `physicianEar`—Physicians and nurses log in to the physician Web Application (`physicianWebApp`) to search and access patient profiles, create and review patient medical records, and prescribe medicine to patients. The physician application is designed to communicate using the Web Service provided in `medrecEar`.
- `startupEar`—The `startupEar` application is a simple Web Application that automatically starts a Web browser and loads a MedRec informational page when you start the installed MedRec domain. This application is not discussed during the development tutorials, but is compiled and deployed as part of the complete MedRec build process.

In the tutorials that follow, all applications will be deployed in a single-server domain. Single-server domains are generally used during the development process for convenience of deployment and testing. Figure 1 shows how each application would be deployed to multiple servers in a production environment.

Figure 1: MedRec Application Suite in a Multiple-Server Domain



Throughout the course of the MedRec tutorials, you create the server instances, build the MedRec applications, and deploy them to the new servers. If you are interested in viewing or using the complete MedRec application before starting the tutorials, you can use the pre-built MedRec domain that is installed with WebLogic Server.

While the MedRec tutorials explain how to develop application components using WebLogic Server tools, they do not describe MedRec's J2EE implementation or explain how to program J2EE components in Java. For more information about MedRec's J2EE architecture and implementation, see the *Avitek Medical Records Architecture Guide*.

How to Use the Tutorials

The MedRec tutorials are designed to be completed in the order they are presented. The sequence of tutorials follows the various stages of J2EE application development, from staging and coding the application, through building and deploying components.

If you choose to skip one or more tutorials, read the Prerequisites section of the tutorial you want to follow. This section identifies steps you need to complete in order to complete the tutorial. In many cases, BEA has provided scripts that help you catch up to a given point in the tutorials. If you follow the tutorials in sequence, you will always meet the prerequisites for the next tutorial.

Tutorial Descriptions

The tutorials are divided into the following sections:

- Configuring Domains and Servers describes how to configure the domains, WebLogic Server instances, and resources required to deploy the MedRec application.
- Building the MedRec Applications describes how to create the development environment for the MedRec tutorials and build application components. The development environment consists of the application directories and associated Ant tasks that help you build and deploy the J2EE applications. Tutorials in this section also describe how to use WebLogic Server tools generate deployment descriptors, package, and deploy J2EE components.
- Moving to Production Mode describes how to take the MedRec application from the development environment into a production environment. Tutorials in this section focus on packaging, deploying, and tuning the MedRec application.

Related Reading

- *Introduction to WebLogic Server and WebLogic Express*
- J2EE API Programming Guides
- *Developing WebLogic Server Applications*
- *Avitek Medical Records Architecture Guide*

Configuring Domains and Servers

Tutorial 1: Creating a WebLogic Domain and Server Instance for Development

In this tutorial you use the WebLogic Server Configuration Wizard to create a domain and server necessary to deploy and run the MedRec applications.The tutorial also shows you how to start the server.

The Configuration Wizard asks for information about the domain you want to create based on the configuration template you select, and then creates a `config.xml` file for the domain based on your responses. The Configuration Wizard also creates startup scripts for the server instances in the domain, and other helper files and directories to help you start and use the new domain and its servers. You will work with these scripts and directories in later tutorials.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Make sure WebLogic Server 8.1 and the server samples are installed on your computer.
- Read "Overview of the Avitek Medical Records Development Tutorials."

Procedure

To create the MedRec domain and the WebLogic Server instance to which you will deploy MedRec, follow these steps. You will use the domain and server in later tutorials.

- Step 1: Create the MedRec domain and MedRec server.
- Step 2: Edit the server startup script.
- Step 3: Start the MedRec server.

Step 1: Create the MedRec domain and MedRec server.

The MedRec domain includes one server that will host the MedRec back-end services, the MedRec Administration application, and the Patient application (both applications are Web applications). As you create the domain and server, click Next at the end of each step to continue to the next step in the procedure.

1. Launch the Configuration Wizard:

   Start→Programs→BEA WebLogic Platform 8.1→Configuration Wizard

2. In the Create or Extend a Configuration window, select Create a new WebLogic configuration.
3. In the Select a Configuration Template window, select Basic WebLogic Server Domain.

   You select the Basic WebLogic Server Domain template instead of the Avitek Medical Records Sample Domain template because this tutorial is designed to show you how to create an application from the very beginning. The Avitek

Medical Records Sample Domain template includes configuration settings for the sample domain which would enable you to skip some configuration steps.

4. In the Choose Express or Custom Configuration window, select Custom.
5. In the Configure the Administration Server window, enter or select:
   - `MedRecServer` for Name.
   - `127.0.0.1` for Listen Address.
   - `7101` for Listen Port. If necessary, enter a different value to avoid network communication conflicts with other server instances, such as the Examples server. The port must be dedicated TCP/IP port for the Administration Server. The port number can be any integer from 1 to 65535.
   - The SSL Enabled check box.
   - `7102` for SSL Listen Port. If necessary, enter a different value to avoid network communication conflicts with other server instances, such as the Examples server. The port must be dedicated TCP/IP port and cannot be the same as the Server Listen Port. The port number can be any integer from 1 to 65535.
6. In the following windows, select No:
   - Managed Servers, Clusters, and Machines Options
   - Database (JDBC) Options
   - Messaging (JMS) Options
7. In the Configure Administrative Username and Password window, enter or select:
   - `weblogic` for Name
   - `weblogic` for Password
   - `No` for Configure additional users, groups, and global roles

   You use this username and password when you boot the server and log in to the Administration Console.

   Note: In a production environment the user name and password should not be the same.
8. In the Configure Windows Options window, select:
   - `Yes` for Create Start Menu

- `No` for Install Administrative Server as Service

9. In the Build Start Menu Entries window, accept the defaults.
10. In the Configure Server Start Mode and Java SDK window, select:
    - `Development Mode` for WebLogic Configuration Startup Mode
    - `Sun SDK 1.4.1_XX` for Java SDK Selection

    The Sun SDK is the default choice for Development mode. You can select either the Sun SDK or the JRockit SDK. The Sun SDK offers faster startup times, where as the JRockit SDK offers faster runtime performance on Intel architectures.
11. In the Create WebLogic Configuration window:
    a. Enter `MedRecDomain` as the Configuration Name.
    b. Click Create to create the MedRec domain in the folder displayed in Configuration Location. When the Configuration Wizard finishes creating the domain, the WebLogic Configuration Created Successfully message is displayed.
    c. Click Exit or Done to close the Configuration Wizard.

Step 2: Edit the server startup script.

The MedRec application suite uses log4j for logging application messages. You must copy the log4j properties file from the pre-configured MedRec domain and identify it using a startup option in MedRecServer startup script. For Web Services, you must also identify the `.wsdl` and the incoming directory for XML files. To complete these steps:

1. Copy the log4j properties file from the pre-configured MedRec domain to the new domain you just created. For example, in a command-line shell, enter:

   ```
   copy c:\bea\weblogic81\samples\domains\medrec\log4j.properties
   c:\bea\user_projects\domains\MedRecDomain
   ```
2. Open the `startWebLogic.cmd` script for your new domain in a text editor. For example:

   ```
   notepad
   c:\bea\user_projects\domains\MedRecDomain\startWebLogic.cmd
   ```

3. Find the following line in the `startWebLogic.cmd` script:

   ```
   set JAVA_VENDOR=Sun
   ```

4. Add the following line immediately after the "set JAVA_VENDOR" line:

   ```
   Set JAVA_OPTIONS=-Dlog4j.config=log4j.properties
   -Dcom.bea.medrec.xml.incoming=incoming
   -Dphys.app.wsdl.url=http://127.0.0.1:7101/ws_medrec/MedRecWebSe
   rvices?WSDL
   ```

5. Save the file and exit your text editor.

Step 3: Start the MedRec server.

From the Start menu:

Start→Programs→BEA WebLogic Platform 8.1→User Projects→MedRecDomain→Start Server

From a script:

1. In a command-line shell, go to the root directory of the MedRec domain, typically `c:\bea\user_projects\domains\MedRecDomain`. For example, from the `c:\` prompt, enter:

   ```
   cd bea\user_projects\domains\MedRecDomain
   ```

2. Invoke the `startWebLogic.cmd` script to start the MedRec server:

   Windows: `startWebLogic.cmd`

   UNIX: `startWeblogic.sh`

Best Practices

- Use the Configuration Wizard to create and configure domains. The Configuration Wizard creates the necessary configuration file (`config.xml`), directory structure, and startup scripts for each new domain.
- Create domain directories outside the WebLogic Server program files. It is best not to mix application files with the application server files. By default, the Configuration Wizard creates domain directories in *`bea_home`*`\user_projects\domains` directory, typically

`c:\bea\user_projects\domains`, which is parallel to the directory in which WebLogic Server program files are stored, typically `c:\bea\weblogic81`.

The Big Picture

This tutorial is the basis for setting up your development environment. Before you can deploy applications to a server, you must first configure the domains and servers to which you want to deploy the applications. In this tutorial, you created the MedRec domain, which includes one server to host the MedRec applications. You use this domain for most tutorials.

Related Reading

- *Creating Domains and Servers Using the Configuration Wizard* in *Configuring and Managing WebLogic Server*
- Starting and Stopping Servers: Quick Reference in *Configuring and Managing WebLogic Server*
- Starting Administration Servers in the *Administration Console Online Help*

1 Configuring Domains and Servers

Tutorial 2: Starting the PointBase Development Database

This tutorial describes how to start the PointBase database management system so that the MedRec application can use it to store application data.

In particular, the tutorial shows how to:

- Start the PointBase database.
- Use the PointBase console to view the tables in the database used by the MedRec application.

Note: The installation of PointBase shipped with WebLogic Server is already set up with the database tables and data used by the MedRec application. For information on viewing the already-created tables, see Step 2: Use the PointBase console to view the MedRec tables and data.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial, create the MedRec domain and MedRec server instance. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development".

Procedure

Follow these steps to start and use PointBase:

Step 1: Start the PointBase database.

1. Open a command prompt window.
2. Change to the PointBase tools directory:

   ```
   cd c:\bea\weblogic81\common\eval\pointbase\tools
   ```

3. Start the PointBase database by executing the following command:

   ```
   startPointBase.cmd
   ```

4. Leave this command window open for as long as you want the PointBase database running. If you close the window, the PointBase database will shut down.

Step 2: Use the PointBase console to view the MedRec tables and data.

The installation of PointBase shipped with WebLogic Server is already set up with the database tables and data used by the MedRec application. To view these already-created tables, use the PointBase Console.

Note: You must start the PointBase database before you can start the PointBase console. See Step 1: Start the PointBase database.

1. Launch the PointBase console:

   From the Start menu:

   Start→Programs→BEA WebLogic Platform 8.1→Examples→WebLogic Server Examples→PointBase Console

   From a script:

   a. In a command-line shell, go to the *bea_home*`\weblogic81\common\eval\pointbase\tools` directory where *bea_home* is the main BEA home directory, typically `c:\bea`. For example, enter the following command:

      ```
      cd c:\bea\weblogic81\common\eval\pointbase\tools
      ```

   b. Invoke the `startPointBaseConsole.cmd` command to launch the PointBase console:

      ```
      startPointBaseConsole.cmd
      ```

      This command also sets the CLASSPATH to find the PointBase JAR files.

2. In the Driver field, enter `com.pointbase.jdbc.jdbcUniversalDriver`.
3. In the URL field, enter `jdbc:pointbase:server://localhost/demo`.
4. In the User field, enter `MedRec`.
5. In the Password field, enter `MedRec`.
6. Click OK.
7. In the left pane, expand Schemas→MedRec.
8. Browse the tables, triggers, views, and procedures that make up the MedRec database.

Best Practices

Use the scripts in the PointBase tools directory to start the database and invoke its console. See:

```
c:\bea\weblogic81\common\eval\pointbase\tools
```

The Big Picture

The MedRec application uses the PointBase database management system:

- To store information about patients, physicians, and administrators who manage the workflow of the MedRec application.
- As the JMS JDBC store that contains persistent JMS messages.

Patient, Physician, and Administrator Data

The MedRec application uses container-managed entity EJBs to automatically persist information about patients, physicians, and administrators in the PointBase database. The following table lists these entity EJBs and the PointBase tables in which the information is persisted.

Table 1: Relationship Between MedRec Entity EJBs and PointBase Tables

Entity EJB	Application That Uses the EJB	Corresponding PointBase Table	Description
AdminEJB	Administration	ADMIN	Information about the administrators that manage the workflow of the MedRec application. Administrators handle patient requests.
AddressEJB	Administration, Patient	ADDRESS	Used by the PATIENT, PHYSICIAN, and ADMIN tables to store their respective addresses.
PatientEJB	Administration, Patient	PATIENT	Information about patients, such as name, address reference to the ADDRESS table, SSN, and so on.
PhysicianEJB	Administration	PHYSICIAN	Information about physicians, such as name, address reference to the ADDRESS table, phone, and email.

Table 1: Relationship Between MedRec Entity EJBs and PointBase Tables

Entity EJB	Application That Uses the EJB	Corresponding PointBase Table	Description
PrescriptionEJB	Patient	PRESCRIPTION	Describes a prescription, such as the prescribed drug, the dosage, frequency, instructions, and so on. Also includes the patient ID, the ID of the prescribing physician, and the particular visit that instigated the prescription.
RecordEJB	Patient	RECORD	Describes a single patient visit to a physician. Includes the patient ID, the physician ID, the date, the symptoms, diagnosis, and the vital signs of the patient.
UserEJB	Administration, Patient, Physician	USER	Lists all users (patients, physicians, and administrators) who are authorized to log into the MedRec application. After a user is authenticated, the application retrieves additional information from the appropriate table (PATIENT, PHYSICIAN, OR ADMIN).
VitalSignsEJB	Patient	VITALSIGNS	Describes the vital signs of a patient for a particular visit. Vital signs include temperature, blood pressure, height, weight, and so on.

Persistent JMS Message Storage

The MedRec application uses persistent JMS messaging, which means that any JMS messages that are put in a queue are also stored in a database so that the messages can be retrieved in case a problem occurs (such as a server crash) before the message-driven bean is able to process them. The messages are stored in the following two PointBase tables:

- MEDRECJMSSTATE
- MEDRECJMSSTORE

These tables are generated automatically when you create the JMS JDBC store using the Administration Console and are used internally by JMS.

Related Reading

- *PointBase Console Guide* at http://e-docs.bea.com/wls/docs81/pdf/pbconsole.pdf
- *PointBase Developer's Guide* at http://e-docs.bea.com/wls/docs81/pdf/pbdeveloper.pdf
- *PointBase System Guide* at http://e-docs.bea.com/wls/docs81/pdf/pbsystem.pdf
- *Understanding Enterprise Java Beans (EJB)* at http://e-docs.bea.com/wls/docs81/ejb/understanding.html
- *Designing Enterprise Java Beans* at http://e-docs.bea.com/wls/docs81/ejb/design_best_practices.html
- *Entity EJBs* at http://e-docs.bea.com/wls/docs81/ejb/entity.html

1 Configuring Domains and Servers

Tutorial 3: Setting Up WebLogic Server Resources for the MedRec Server

This tutorial describes how to set up the WebLogic Server resources required to deploy and run the MedRec application. The tutorial sets up resources for the MedRec server. These resources include:

- Java Database Connectivity (JDBC) connection pools and data sources for the PointBase database management system
- Java Message Service (JMS) persistent store, JMS server, queue, and connection factory
- JavaMail mail sessions

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create the MedRec domain and MedRec server, and start the MedRec server. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development".
- Start the PointBase database management system. See Tutorial 2: Starting the PointBase Development Database.

Procedure

Follow these steps to configure WebLogic Server resources for the MedRec server:

- Step 1: Invoke the Administration Console for the MedRec server in your browser.
- Step 2: Create the JDBC connection pools.
- Step 3: Create a JDBC DataSource.
- Step 4: Create a JMS JDBC store.
- Step 5: Create a JMS server.
- Step 6: Create the JMS queues.
- Step 7: Create a JMS connection factory.
- Step 8: Add email capabilities to the MedRec application.
- Step 9: Configure the MedRec Sample Authenticator.

Step 1: Invoke the Administration Console for the MedRec server in your browser.

You use the Administration Console to create the WebLogic Server resources used by the MedRec application suite.

1. Enter the following URL in your browser:

```
http://127.0.0.1:7101/console
```

2. Enter `weblogic` as the username and password, then click Sign In.

Step 2: Create the JDBC connection pools.

A JDBC connection pool describes how to physically connect to a database, in this case a PointBase database. This procedure describes how to create two JDBC connection pools: the first uses an XA JDBC driver and the second one does not.

Typically you always use an XA JDBC driver when creating a connection pool. However, because JMS JDBC stores do not support XA resource drivers (WebLogic JMS implements its own XA resource), a second non-XA connection pool is needed. Later procedures show how to associate the XA connection pool to a JDBC DataSource and the non-XA connection pool to a JMS JDBC store.

1. In the left pane of the Administration Console, expand Services→JDBC.
2. Click Connection Pools.
3. In the right pane, click Configure a new JDBC Connection Pool.
4. Select `PointBase` as the Database Type.
5. Select `PointBase's Driver (Type 4XA) Versions:4.X` as the Database Driver.
6. Click Continue.
7. In the Name field, enter `MedRecPool-PointBase-XA`.
8. In the Database Name field, enter `demo`.
9. Accept `localhost` as the Host Name of the computer that is hosting PointBase.
10. In the Port field, enter `9092`.
11. In the Database User Name field, enter `medrec`.
12. In the Password and Confirm Password fields, enter `medrec`.
13. Click Continue.
14. Ensure that the information to test the connection to the PointBase database is correct, then click Test Driver Configuration.

Note: Be sure you have started PointBase, or the test of its driver configuration will fail. For details, see Tutorial 2: Starting the PointBase Development Database.

15. After verifying that the connection succeeded, click Create and Deploy.
16. Click Configure a new JDBC Connection Pool.
17. Select `PointBase` as the Database Type.
18. Select `PointBase's Driver (Type 4) Versions:4.X` as the Database Driver.
19. Click Continue.
20. In the Name field, enter `MedRecPool-PointBase`.
21. In the Database Name field, enter `demo`.
22. Accept `localhost` as the Host Name of the computer that is hosting PointBase.
23. In the Port field, enter `9092`.
24. In the Database User Name field, enter `medrec`.
25. In the Password and Confirm Password fields, enter `medrec`.
26. Click Continue.
27. Ensure that the information to test the connection to the PointBase database is correct, then click Test Driver Configuration.
28. After verifying that the connection succeeded, click Create and Deploy.

Step 3: Create a JDBC DataSource.

Client and server-side JDBC applications obtain a DBMS connection through a DataSource. A DataSource is an interface between an application and the JDBC connection pool. This DataSources uses the XA connection pool you created in Step 2: Create the JDBC connection pools.

1. In the left pane of the Administration Console, expand Services→JDBC.
2. Click Data Sources.
3. In the right pane, click Configure a new JDBC Data Source.

4. In the Name field, enter `MedRecTxDataSource`.
5. In the JNDI Name field, enter `MedRecTxDataSource`.
6. Select the Honor Global Transactions checkbox.
7. Click Continue.
8. In the Pool Name list box, select `MedRecPool-PointBase-XA`.
9. Click Continue.
10. Ensure that `MedRecServer` is selected as the server on which you want to deploy this data source.
11. Click Create.

Step 4: Create a JMS JDBC store.

JMS stores are used to store persistent messages. This JMS JDBC store uses the non-XA connection pool you created Step 2: Create the JDBC connection pools.

1. In the left pane of the Administration Console, expand Services→JMS.
2. Click Stores.
3. In the right pane, click Configure a new JMS JDBC Store.
4. In the Name field, enter `MedRecJMSJDBCStore`.
5. In the Connection Pool list box, select `MedRecPool-PointBase`.
6. In the Prefix Name field, enter `MedRec`.
7. Click Create.

Step 5: Create a JMS server.

JMS servers host the queue and topic destinations used by JMS clients. To persistently store messages in destinations, the JMS server must be configured with a JMS store.

1. In the left pane of the Administration Console, expand Services→JMS.
2. Click Servers.

3. In the right pane, click Configure a new JMS Server.
4. In the Name field, enter `MedRecJMSServer`.
5. In the Persistent Store list box, select `MedRecJMSJDBCStore`.
6. Click Create.
7. In the Target list box, select `MedRecServer`.
8. Click Apply.

Step 6: Create the JMS queues.

JMS queues are based on the point-to-point (PTP) messaging model, which enables the delivery of a message to exactly one recipient. A queue sender (producer) sends a message to a specific queue. A queue receiver (consumer) receives messages from a specific queue.

The following procedure describes how to create three JMS queues, which are used by message-driven beans for registering new users of the MedRec application, handling email, and uploading XML files.

1. In the left pane of the Administration Console, expand Services→ JMS→Servers→MedRecJMSSserver.
2. Create the queue for the registration message-driven bean:
   a. In the left pane, click Destinations.
   b. In the right pane, click Configure a new JMS Queue.
   c. In the Name field, enter `jms/REGISTRATION_MDB_QUEUE`.
   d. In the JNDI Name field, enter `jms/REGISTRATION_MDB_QUEUE`.
   e. Click Create.
3. Create the queue for the email message-driven bean:
   a. In the left pane, click Destinations.
   b. In the right pane, click Configure a new JMS Queue.
   c. In the Name field, enter `jms/MAIL_MDB_QUEUE`.

d. In the JNDI Name field, enter `jms/MAIL_MDB_QUEUE`.

e. Click Create.

4. Create the queue for the XML upload message-driven bean:

a. In the left pane, click Destinations.

b. In the right pane, click Configure a new JMS Queue.

c. In the Name field, enter `jms/XML_UPLOAD_MDB_QUEUE`.

d. In the JNDI Name field, enter `jms/XML_UPLOAD_MDB_QUEUE`.

e. Click Create.

Step 7: Create a JMS connection factory.

JMS clients use JMS connection factories to create a connection to a WebLogic Server instance. JMS client messaging requests to a particular destination are routed through their connection's host WebLogic Server to the WebLogic Server hosting the JMS server destination. JMS connection factories are also used to configure the defaults for the JMS clients that use them.

1. In the left pane of the Administration Console, expand Services→JMS.
2. Click Connection Factories.
3. In the right pane, click Configure a new JMS Connection Factory.
4. In the Name field, enter `jms/MedRecQueueConnectionFactory`.
5. In the JNDI Name field, enter `jms/MedRecQueueConnectionFactory`.
6. Click Create.
7. In the Targets box, select `MedRecServer`.
8. Click Apply.
9. Select the Configuration→Transactions tab.
10. Select the XA Connection Factory Enabled check box.
11. Click Apply.

Step 8: Add email capabilities to the MedRec application.

WebLogic Server includes the JavaMail API version 1.1.3 reference implementation from Sun Microsystems. Using the JavaMail API, you can add email capabilities to your WebLogic Server applications. To configure JavaMail for use in WebLogic Server, you create a Mail Session in the WebLogic Server Administration Console. A mail session allows server-side components and applications to access JavaMail services with JNDI, using Session properties that you preconfigure.

1. In the left pane of the Administration Console, expand Services→Mail.
2. In the right pane, click Configure a new Mail Session.
3. In the Name field, enter `mail/MedRecMailSession`.
4. In the JNDIName field, enter `mail/MedRecMailSession`.
5. In the Properties text box, enter values for the `mail.user` and `mail.host` properties.

   For example, if you want any email generated by the MedRec application to be sent to you, and your email address is `joe@mail.mycompany.com`, enter:

   ```
   mail.user=joe;mail.host=mail.mycompany.com
   ```

6. Click Create.
7. In the Targets box, select `MedRecServer`.
8. Click Apply.

Step 9: Configure the MedRec Sample Authenticator.

The MedRec Sample Authenticator retrieves login credentials from the configured PointBase RDBMS for a given username. Within the provider, passwords are validated, and if correct, the user's group associations are retrieved.

1. In the left pane of the Administration Console, expand the Security→Realms→myrealm→Providers node.
2. Select the Authentication node under the Providers node.
3. In the right pane, select Configure a New MedRec Sample Authenticator.
4. On the General tab in the right pane:

- In the Name field, enter MedRecSampleAuthenticator.
- From the Control Flag menu, select SUFFICIENT.
- Click Create to create the new authenticator.

The SUFFICIENT control flag indicates that the LoginModule does not need to succeed. If it does succeed, control is returned to the application. However, if it does not succeed, the server tries other configured authentication providers.

5. In the left pane of the Administration Console, expand Security→Realms→myrealm→Providers→Authentication.
6. In the left pane of the Administration Console, click DefaultAuthenticator.
7. In the General tab in the right pane, select SUFFICIENT from the Control Flag menu.
8. Click Apply.
9. Because WebLogic Server cycles through available Authentication providers, reorder the provider list so that the PointBase database is not queried each time a login is attempted (for example, each time you log into the Administration Console in subsequent tutorials).

   In the left pane of the Administration Console, select Security→Realms→myrealm→Providers→Authentication.
10. In the right pane of the Console, click Re-Order the Configured Authentication Providers.
11. Use the arrows in the Configured Providers list to define the following order of authentication providers:

    a. DefaultAuthenticator

    b. MedRec Samples Authenticator

    c. DefaultIdentityAsserter
12. Click Apply.

Best Practices

- When you create a JDBC DataSource, be sure you enable global transaction support (by selecting the Honor Global Transactions box) so that your application can support transaction services.

  The MedRec application uses a DataSource with transactions enabled (a `TxDataSource` object) when persisting application data to the database using entity beans.

- You typically want to use an XA JDBC driver when creating a JDBC connection pool.

- JMS JDBC stores do not support XA resource drivers as WebLogic JMS implements its own XA resource. Therefore, do not associate a connection pool that uses an XA JDBC driver with a JMS JDBC store.

- In most cases, to avoid unnecessary JMS request routing, the JMS connection factory should be targeted to the same WebLogic Server instance as the JMS server.

- When configuring the persistent JMS store, you can persist JMS messages to a directory on the file system (called JMS file store) or to a database using JDBC (called JMS JDBC database store).

  If you want better performance and simpler configuration, BEA recommends you persist JMS messages to the file system. If you want to store your persistent messages in a remote database rather than on the JMS server's host machine, BEA recommends you use a JDBC JMS store.

- Always configure quotas for WebLogic JMS servers. JMS quotas prevent too many messages from overflowing server memory. In addition, consider configuring message paging, as persistent and non-persistent messages consume server memory unless paging is enabled.

The Big Picture

The MedRec application uses JMS to create a new patient record. The asynchronous nature of JMS allows the task to be queued and completed later while the user continues with another task.

After the user clicks Create on the Web page to register a new patient, a JMS message is created and put on the `REGISTRATION_MDB_QUEUE` JMS queue. The `RegistrationEJB` message-driven bean takes the message off the queue and persists the new patient data to the database using an instance of the `PatientEJB` entity bean. The `PatientEJB` entity bean uses the JDBC DataSource to connect to the PointBase database.

The MedRec application uses other entity beans to persist additional data to the database; for details, see "Patient, Physician, and Administrator Data" on page 1-4.

The MedRec application uses persistent JMS messaging, which means that the new patient JMS messages that are put on the queue are also stored in a PointBase database so that the messages can be retrieved in case a problem occurs (such as a server crash) before the message-driven bean is able to process them. The application uses the JMS JDBC store to connect to and to update the JMS tables in the PointBase database.

Related Reading

- *Introduction to WebLogic JDBC* at http://e-docs.bea.com/wls/docs81/jdbc/intro.html
- *JDBC Connection Pools* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html
- JMS Topic Page at http://e-docs.bea.com/wls/docs81/messaging.html
- *WebLogic JMS Fundamentals* at http://e-docs.bea.com/wls/docs81/jms/fund.html
- *Configuring JMS* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jms_config.html
- *Using JavaMail with WebLogic Server Applications* at http://e-docs.bea.com/wls/docs81/programming/topics.html#topics003

1 Configuring Domains and Servers

Tutorial 4: Using WebLogic Server Development Mode

This tutorial describes how to set up and use the new MedRec server instance in development mode. WebLogic Server provides two distinct server modes—development mode and production mode—that affect default configuration values and subsystem behavior for all server instances in a domain.

Development mode enables you to use the demonstration trusted CA certificates for security, and also allows you to deploy the MedRec applications directly from a development environment. (You will create the development environment in the next set of tutorials). For these reasons, you should always use development mode when building or testing your own applications.

Note: Because newly installed WebLogic Server instances use development mode by default, the steps in this tutorial are not strictly required. However, later tutorials that describe how to move from a development to a production environment depend on the changes you make now.

The tutorial includes the following sections:

- Prerequisites
- Procedure

- Best Practices
- The Big Picture
- Related Reading

Prerequisites

Before starting this tutorial, create the MedRec server domain. See Tutorial 1: Creating a WebLogic Domain and Server Instance for Development. You will modify the server start script that was created during that tutorial.

Procedure

Follow these steps to put MedRec server in development mode:

- Step 1: Shut down the MedRec server (if currently running).
- Step 2: Edit the server startup file.
- Step 3: Restart the server and verify development mode.

Step 1: Shut down the MedRec server (if currently running).

You must shut down the MedRec server because you edit its start script to explicitly place the server in development mode.

If the server is not currently running, go to Step 2: Edit the server startup file.

1. Invoke the Administration Console for MedRecServer by entering the following URL in your browser:

   ```
   http://127.0.0.1:7101/console
   ```

2. Enter `weblogic` as the username and password, then click Sign In.
3. In the left pane, open the Servers node.
4. Right-click MedRecServer and select Start/Stop This Server.
5. In the right pane, click Graceful shutdown of this server.

6. Click Yes.

Step 2: Edit the server startup file.

Development mode (or production mode) is set for all servers in a given domain by supplying a command line option to the domain's Administration Server. Because the MedRec tutorials use two standalone servers in separate domains, you must edit each server's startup script to add the command line option.

1. In a command-line shell, move to the root directory of the MedRec domain:

   ```
   cd c:\bea\user_projects\domains\MedRecDomain
   ```

2. Open the `startWebLogic.cmd` or `startWebLogic.sh` script in a text editor:

   ```
   notepad startWebLogic.cmd
   ```

3. Look for the `PRODUCTION_MODE` script variable:

   ```
   set PRODUCTION_MODE=
   ```

4. Add "false" to the value of the `PRODUCTION_MODE` variable to ensure the server starts in development mode:

   ```
   set PRODUCTION_MODE=false
   ```

5. Save your changes and exit the text editor.

Step 3: Restart the server and verify development mode.

After editing the server start script, reboot the server to ensure that it starts up in development mode:

1. Start the MedRec server by executing its startup script:

   ```
   C:\bea\user_projects\domains\MedRecDomain\startWebLogic.cmd
   ```

2. Observe the server startup message to determine the startup mode. The following line indicates that the server is using development mode:

   ```
   <Jul 10, 2003 5:40:01 PM PDT> <Notice> <WebLogicServer>
   <BEA-000331> <Started WebLogic Admin Server "MedRecServer" for
   domain "MedRecDomain" running in Development Mode>
   ```

Best Practices

- Use development mode in a WebLogic Server domain to:
  - Develop, modify, and test applications in a development environment.
  - Enable auto-deployment for applications placed in the `\applications` directory
  - Use demonstration trusted CA certificates for testing security configurations
  - Automatically create a JMS file store directory if needed for an application
- If you start an Administration Server from the command line, or if you use custom startup scripts, use the `weblogic.Server` command-line arguments `-DProductionModeEnabled=true | false` to set the server mode.
- Never use development mode for production-level servers, because development mode relaxes the security constraints for all servers in the domain.

The Big Picture

The MedRec application uses the sample trusted CA certificates installed with WebLogic Server to enable SSL authentication and demonstrate WebLogic Server security features in later tutorials. Development mode allows you to use the sample certificate files when working through later security tutorials.

In the next series of tutorials, you will create a development directory structure for MedRec that shows how to manage source code and compiled code separately when developing Enterprise Applications with WebLogic Server. Development mode allows you to deploy applications directly from the development directory, without having to package applications into `.jar` files or exploded `.jar` directories.

Related Reading

- WebLogic Server Command-Line Reference
- Developing WebLogic Server Applications

Building the MedRec Applications

Tutorial 5: Creating the MedRec Project Directory

This tutorial describes how to create the main project directory that holds the MedRec source files and compiled classes. The tutorial also explains the high-level directory structure and contents for the MedRec application suite components.

Tutorials that follow provide more detail about the development directory structure and WebLogic Server ant tasks that help you easily build and deploy Enterprise Applications and their subcomponents—Web applications, EJBs, and Web services.

This tutorial includes the following sections:

Prerequisites

Before starting this tutorial, create the MedRec domain and MedRec server. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development".

Procedure

Follow these steps to create the source directory structure for the MedRec application suite:

- Step 1: Create the tutorial project directory.
- Step 2: Unpack the project subdirectories.
- Step 3: Verify the project directory contents.
- Step 4: Verify the source directory contents.
- Step 5: Edit the \src\medrec.properties file and run substitute.xml.

Step 1: Create the tutorial project directory.

Begin by creating a top-level project directory in which you will store source and output files for the MedRec Enterprise Applications and client programs. Name the directory `medrec_tutorial`:

```
mkdir c:\medrec_tutorial
```

Step 2: Unpack the project subdirectories.

BEA provides a `.zip` file that contains the source files and build subdirectories needed to complete the MedRec tutorials. To populate your project directory with the necessary files and directories:

1. Download the `medrec_tutorial.zip` file from http://edocs.bea.com/wls/docs81/medrec_tutorial.zip. Save the downloaded file to the `c:\medrec_tutorial` directory.
2. Set your command shell environment with the MedRecDomain environment script:

```
c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
```

3. Move to the MedRec project directory and unpack the downloaded `.zip` file:

```
cd c:\medrec_tutorial
jar xvf medrec_tutorial.zip
```

Step 3: Verify the project directory contents.

Verify that the following files and subdirectories were created:

```
dir

 Directory of C:\medrec_tutorial

02/13/2003  08:59a      <DIR>          .
02/13/2003  08:59a      <DIR>          ..
02/13/2003  08:59a      <DIR>          build
02/13/2003  08:59a      <DIR>          dist
02/13/2003  08:59a      <DIR>          lib
02/13/2003  08:59a           2,782,963 medrec_tutorial.zip
02/13/2003  08:59a      <DIR>          META-INF
02/13/2003  08:59a                4,505 README.TXT
02/13/2003  08:59a      <DIR>          src
02/13/2003  08:59a                  299 substitute.xml
```

The `\build` directory contains the compiled classes generated by the various MedRec build scripts. It does not contain editable source files or deployment descriptors, which reside in `\src`. Each subdirectory in `\build` represents the compiled classes for MedRec clients (`\client`) or for a MedRec application (`\medrec`, `\physician`, and `\startup` applications).

If you look at the contents of the `\build` directory, you notice that certain classes have already been built for you. These include utility classes and the MedRec value objects that many applications in the MedRec tutorial require. Having these classes prebuilt allows you to concentrate on compiling Enterprise Applications using the WebLogic `wlcompile` task, as described in Tutorial 7: Compiling Applications Using the Split Development Directory.

The `\src` directory contains the full source for all MedRec applications. You will be working in this directory for most remaining tutorials. Step 4: Verify the source directory contents. describes the subdirectories in `\src`.

\build and \src together represent a WebLogic Server split development directory. You can deploy individual MedRec applications to a development server by targeting an application subdirectory in \build (such as \build\medrecEar) using weblogic.Deployer or the wldeploy ant task described in Tutorial 9: Deploying MedRec from the Development Environment. WebLogic Server locates the necessary deployment descriptors (available in \src) by examining the .beabuild.txt file located in \build subdirectory.

The \dist directory is also an output directory—it will store the archived .ear files or exploded .ear directories created by the wlpackage task in Tutorial 14: Packaging MedRec for Distribution. Right now it contains only the MedRecService.wsdl file which is required to compile parts of the physicianEar application in Tutorial 7: Compiling Applications Using the Split Development Directory. It will eventually store complete, exploded .ear directories for the different MedRec applications. \dist is not considered part of the split development directory structure, because it is not required for compiling or deploying applications during development. It is used only for storing final, completed applications—.ear files or exploded .ear directories—that you generate after completing the development process.

\lib contains precompiled, third-party .jar files that several of the MedRec applications require. This includes supporting .jars for struts and log4j.

Step 4: Verify the source directory contents.

The \src subdirectory contains the full application source for the MedRec applications, and it is the subdirectory in which you will spend the most time during the remaining tutorials. Take a look at the installed \src directory:

```
dir src

 Directory of C:\medrec_tutorial\src

02/13/2003  11:22p      <DIR>          .
02/13/2003  11:22p      <DIR>          ..
02/13/2003  11:22p               1,909 build.xml
02/13/2003  11:22p      <DIR>          clients
02/13/2003  11:22p      <DIR>          common
02/13/2003  11:23p               3,479 medrec.properties
02/13/2003  11:22p      <DIR>          medrecEar
02/13/2003  11:22p      <DIR>          physicianEar
02/13/2003  11:22p      <DIR>          security
02/13/2003  11:22p      <DIR>          startupEar
```

The `build.xml` file in the top level of the `medrec_tutorial` directory is a project-wide build file. It:

- Cleans up previously-built versions of MedRec before compiling
- Builds the contents of each application subdirectory into the `\build` directory by calling each application's `build.xml` file
- Packages each application as an exploded `.ear` file into the `\dist` directory

You will use this project-level `build.xml` before moving the WebLogic Server instance into production mode in Tutorial 13: Compiling the Entire MedRec Project. However, do not try to use it yet—you need to complete the next few tutorials to create the application-level `build.xml` files that this script calls.

The subdirectories of `\src` represent either deployable MedRec applications or MedRec components that are used by those applications:

- `\clients` holds source files for the Java and C# clients of MedRec Web Services.
- `\common` holds source files for Java classes shared between the MedRec Enterprise Applications. These include:
  - Shared constants and JNDI names
  - The ServiceLocator class, used to access MedRec services in the service tier
  - Factories for creating EJBs and JMS connections
  - Value objects, which represent data passed between tiers of the MedRec application
  - Image files and Struts action classes shared across MedRec web components
- `The \medrecEar` and `\physicianEar` subdirectories store the main Enterprise Applications that make up the MedRec application suite. These subdirectories use the WebLogic Server 8.1 Development Directory structure and ant tasks for building and deploying, and are described in detail in the next tutorials.
- `The \security` subdirectory contains the MedRec authentication provider shared across applications.
- `The \startupEar` subdirectory contains the startup class that automatically boots the browser and loads MedRec's main index JSP when you start MedRec on a Windows machine. You do not work directly with this application in the

tutorials that follow. However, the application is compiled as part of the overall MedRec build process.

Step 5: Edit the \src\medrec.properties file and run substitute.xml.

`\src` also contains a `medrec.properties` file that defines property values used by the project-level `build.xml` file, as well as the `build.xml` files used in each applications subdirectory. Follow these instructions to edit the properties file so that it points to your tutorial domain and project directory:

1. Use a text editor to open the properties file:

   ```
   notepad c:\medrec_tutorial\src\medrec.properties
   ```

2. Edit the `wl.home` property to point to your WebLogic Server installation directory (`c:/bea/weblogic81` by default):

   ```
   wl.home=c:/bea/weblogic81
   ```

3. Edit the `port` property, setting the value to `7101`:

   ```
   port=7101
   ```

4. Edit the `medrec.domain.dir` property to point to the MedRecDomain directory you created:

   ```
   medrec.domain.dir=c:/bea/user_projects/domains/MedRecDomain
   ```

5. Edit the `medrec.home` property to point to your new project directory:

   ```
   medrec.home.dir=c:/medrec_tutorial
   ```

6. The remaining property definitions build on `medrec.home` and you do not need to modify them.

7. Save your changes and exit the editor.

8. Go to the `c:\medrec_tutorial` directory and run the `substitute.xml` script:

   ```
   cd c:\medrec_tutorial
   ant -f substitute.xml
   ```

   This script substitutes a variable in the project files with the path to your WebLogic home directory.

Best Practices

- Smaller J2EE projects may not require the nested subdirectories found in the MedRec project directory. For example, a project that produces a single Enterprise application file can have minimal subdirectories such as:
  - `\myProject`—top-level project directory
  - `\myProject\myEarBuild`—output directory for storing compiled and generated files
  - `\myProject\myEarSrc`—source files and editable content for the Enterprise Application

  This minimal directory structure still allows you to develop your application using the WebLogic split development directory structure and ant tasks described in Tutorial 6: Understanding the WebLogic Server Split Directory Structure.

The Big Picture

The MedRec application suite consists of three separate applications for the patient, physician, and administrator user roles. Using a separate application for each user role allows you to distribute each application function across different WebLogic Server instances as needed. For example, the MedRec sample domain (optionally installed with WebLogic Server) deploys all three applications on a single server instance for easy demonstration purposes. The MedRec tutorials also deploy the applications in a single-server domain, which is typical for development environments. However, you can also deploy the MedRec and Physician applications on two different server instances (in separate domains) to illustrate the use of Web Services between the applications.

The MedRec project directory also contains subdirectories for compiling the client applications that access MedRec via Web Services.

Related Reading

- Developing WebLogic Server Applications

- Programming WebLogic Web Services

1 Building the MedRec Applications

Tutorial 6: Understanding the WebLogic Server Split Directory Structure

Several subdirectories in the `medrec_tutorial` project directory—`medrecEar`, `physicianEar`, `startupEar`—use the WebLogic Server split development directory structure for storing source files. The split development directory consists of a directory layout and supporting ant tasks that help you easily build, deploy, and package Enterprise Application files while automatically maintaining classpath dependencies. The directory structure is split because source files and editable deployment descriptors reside in one directory while compiled class files and generated deployment descriptors reside in a separate directory.

The split development directory structure is a valuable tool to use for developing your own applications. Because source files and generated files are kept separate, you can easily integrate your development projects with source control systems. The split development directory also allows you to easily deploy your applications without having to first copy files and stage applications—WebLogic Server automatically uses the contents of both the build and source directories to deploy an application.

This tutorial explains the layout and function of the source directory structure used in the MedRec application suite. The next tutorial describes the build directory structure, which is produced when you compile an Enterprise Application using the `wlcompile` task. The source and build directories together make up a WebLogic split development directory, which you will deploy and package in later tutorials.

This tutorial includes the following sections:

- Prerequisites
- Procedure
- Best Practices
- The Big Picture
- Related Reading

Prerequisites

Before starting this tutorial:

- Create the project directory and unpack the MedRec tutorial source files to it using the instructions in Tutorial 5: Creating the MedRec Project Directory.

Procedure

The following procedure guide you through the source directory structure for the MedRec application suite:

- Step 1: Examine the Enterprise Application directory structure.
- Step 2: Examine the Web Application component directory structure.
- Step 3: Examine the EJB component directory structure.

Step 1: Examine the Enterprise Application directory structure.

The WebLogic split development directory stores source files starting at the Enterprise Application (EAR) level. Even if you are developing only a single Web Application or EJB, you store the relevant component in a top-level directory that represents an Enterprise Application. For example, examine the contents of the `startupEar` subdirectory, which stores the source code for a single Web Application in the MedRec application suite:

```
cd c:\medrec_tutorial\src\startupEar
```

```
dir

 Directory of C:\medrec_tutorial\src\startupEar

01/23/2003  08:43a      <DIR>          .
01/23/2003  08:43a      <DIR>          ..
01/23/2003  08:43a               3,368 build.xml
01/23/2003  08:41a      <DIR>          META-INF
01/23/2003  08:24a      <DIR>          startupWebApp
```

Because this Web Application must be packaged independently of the other MedRec applications, it is placed in its own source directory represented by an EAR file. The `META-INF` subdirectory holds deployment descriptors for the Enterprise Application itself (`application.xml` and optional `weblogic-application.xml` files).

Other source subdirectories in the project directory represent more typical Enterprise Applications having both Web Applications and EJBs, such as `physicianEar`. Move to the `physicianEar` subdirectory and examine its contents:

```
cd c:\medrec_tutorial\src\physicianEar

dir

 Directory of C:\medrec_tutorial\src\physicianEar

01/23/2003  08:43a      <DIR>          .
01/23/2003  08:43a      <DIR>          ..
01/20/2003  05:00p      <DIR>          APP-INF
01/23/2003  08:43a               5,447 build.xml
01/23/2003  08:41a      <DIR>          META-INF
01/23/2003  08:41a      <DIR>          physicianWebApp
01/23/2003  08:24a      <DIR>          sessionEjbs
01/23/2003  08:24a                 218 wlcompile_tutorial.xml
01/23/2003  08:24a                 287 wldeploy_tutorial.xml
01/23/2003  08:24a                 293 wlpackage_tutorial.xml
01/23/2003  08:24a                 393 ws_ejb_client_tutorial.xml
```

As you can see from the directory listing, the Physician application contains both a Web Application component (stored in `physicianWebApp`) and EJB components (stored in `sessionEjbs`). The split development directory structure requires that each EAR component reside in a dedicated source directory. You can name the ear directory and component subdirectories as you see fit, because the `wlcompile` ant task automatically determines the type of component during compilation.

Step 2: Examine the Web Application component directory structure.

The source directory structure allows you to easily manage the different file types that constitute a Web Application. Move to the `physicianWebApp` subdirectory of the `physicianEar` source directory and examine its contents:

```
cd physicianWebApp

dir

 Directory of C:\medrec_tutorial\src\physicianEar\physicianWebApp

02/17/2003  10:01a      <DIR>          .
02/17/2003  10:01a      <DIR>          ..
02/17/2003  09:57a                1,588 Confirmation.jsp
02/17/2003  09:57a                4,538 CreateRx.jsp
02/17/2003  09:57a                9,836 CreateVisit.jsp
02/17/2003  09:57a                2,107 Error.jsp
02/17/2003  09:57a                2,837 Login.jsp
02/17/2003  09:57a                3,770 PatientHeader.jsp
02/17/2003  09:57a                1,940 PhysicianHeader.jsp
02/17/2003  09:57a                2,927 Search.jsp
02/17/2003  09:57a                3,338 SearchResults.jsp
02/17/2003  09:57a                2,840 stylesheet.css
02/17/2003  09:57a                3,549 ViewProfile.jsp
02/17/2003  09:57a                6,443 ViewRecord.jsp
02/17/2003  09:57a                4,846 ViewRecords.jsp
02/17/2003  10:01a      <DIR>          WEB-INF
```

The top level of the Web Application subdirectory contains the JSPs that make up the application. You could also store `.html` files or other static content such as image files here, but it is less cumbersome to store such content in a dedicated subdirectory like `\images`.

Java source files for Web Application components, such as Servlets or supporting utility classes, are stored in package directories under the component's `WEB-INF\src` subdirectory. For example, a utility class for the Physician Web Application is stored in `C:\medrec_tutorial\src\physicianEar\physicianWebApp\WEB-INF\src\com\bea\medrec\utils\PhysConstants.java`.

The `wlcompile` task automatically compiles the contents of the `WEB-INF\src` subdirectory into the `WEB-INF\classes` subdirectory of application's output directory, so that all components of the Web Application can access those classes.

The WEB-INF subdirectory also stores deployment descriptors for the Web Application component (web.xml and the optional weblogic.xml).

Step 3: Examine the EJB component directory structure.

Java source files for EJB components are stored in subdirectories that reflect the EJB's package structure. For example, the source for the Physician Application's session EJB is stored in C:\medrec_tutorial\src\physicianEar\physSessionEjbs\com\bea\medrec\controller\PhysicianSessionEJB.ejb.

Deployment descriptors for EJB components (such as ejb-jar.xml and the optional weblogic-ejb-jar.xml) can be stored in the component's META-INF subdirectory. However, if you look at the physSessionEjbs subdirectory, you will notice there is no META-INF subdirectory. This is because all EJBs in the MedRec application suite use ejbgen tags in their JavaDoc comments, rather than defining them in deployment descriptor files. The wlcompile ant task uses these tags to generate the EJB deployment descriptors automatically when you compile the application.

Best Practices

- Use the same source directory structure with your own J2EE application projects, so you can utilize the WebLogic Server build scripts to compile and deploy your applications. The following summarizes the contents of a simple source directory that follows the WebLogic split development directory structure format:

  ```
  \myProject
  \myProject\myEar
  \myProject\myEar\META-INF\application.xml
  \myProject\myEar\myEjb\com\*\*.java
  \myProject\myEar\myWebApp\*.jsp
  \myProject\myEar\myWebApp\WEB-INF\web.xml
  \myProject\myEar\myWebApp\WEB-INF\src\com\*\*.java
  ```

- Use a source control system to manage the files in the source directory hierarchy. The source directory contains your working files—Java files and

deployment descriptors—and should be regularly backed up to maintain a history of your development project.

- Never store user-generated files in the `\build` directory. The `\build` directory is intended to store only compiled classes for your J2EE applications. You should be able to rebuild the entire `\build` directory simply by recompiling your application.
- You can store deployment descriptor files either in the top level of a J2EE component subdirectory, or in the customary J2EE subdirectory for the component's descriptor files—`\myWebApp\WEB-INF` or `myEjb\META-INF`.

The Big Picture

The MedRec application suite uses three split development directories to hold the source for the `medrecEar`, `physicianEar`, and `startupEar` applications. Utility classes shared among these applications reside in a dedicated directory, `common`, with a custom build script that does not use the split directory structure. Security components are also staged in a custom build directory.

The top-level `build.xml` file iterates through the MedRec source directories and coordinates building all of the components at once.

Although the `wlcompile` task automatically manages most component dependencies during a build, certain split development directories, such as the `medrecEar` and `physicianEar` subdirectories, hard-code the build order to enforce dependencies. The source directory structure that you created during the tutorial contains intermediate build steps, which allow you to focus on using the new WebLogic Server ant tasks without worrying about the dependencies.

Related Reading

- Developing WebLogic Server Applications
- Developing Web Applications for WebLogic Server
- Programming WebLogic Server Enterprise JavaBeans

1 Building the MedRec Applications

Tutorial 7: Compiling Applications Using the Split Development Directory

This tutorial explains how to compile Enterprise Application source files using the `wlcompile` ant task. `wlcompile` works with a WebLogic split development directory structure to produce a build or output directory, which contains the compiled Java classes. The build directory and the source directory described in Tutorial 6: Understanding the WebLogic Server Split Directory Structure constitute a deployable application in WebLogic Server.

Later tutorials explain how to use other WebLogic Server ant tasks that work with the split development directory to perform other application building tasks such as:

- Packaging files from the source and build directories into an EAR file or expanded EAR directory
- Deploying applications

This tutorial includes the following sections:

- The Big Picture
- Related Reading

Prerequisites

Before starting this tutorial:

- Create the project directory and copy over the MedRec source files and output directories using the instructions in Tutorial 5: Creating the MedRec Project Directory.
- Read the instructions in Tutorial 6: Understanding the WebLogic Server Split Directory Structure to understand the organization of source files in the WebLogic Server split development directory.

Procedure

Follow these steps to use the `wlcompile` task with a split development directory in the MedRec application suite:

- Step 1: Create the build.xml file.
- Step 2: Compile the application.
- Step 3: Examine the output files.

Step 1: Create the build.xml file.

Storing your source files using the WebLogic split development directory structure simplifies the `build.xml` file required to compile your applications. For most Enterprise Applications, a simple script of several lines is adequate to compile all modules—the `wlcompile` task automatically determines the modules used in the application and maintains classpath dependencies accordingly.

1. To see how `wlcompile` works, create a simple XML file to compile the Physician application. First move to the `physicianEar` subdirectory in the MedRec project directory:

```
cd c:\medrec_tutorial\src\physicianEar
```

The top-level of `physicianEar` contains subdirectories for the Web Application and EJB components that form the Enterprise Application. You will store the XML file here as well.

2. Use a text editor to create a new `mybuild.xml` file in the `physicianEar` directory:

   ```
   notepad mybuild.xml
   ```

   Note: If you do not want to enter the `build.xml` file manually, copy the file `wlcompile_tutorial.xml` file to the new file name, `mybuild.xml`. Then follow along to understand the file contents.

3. Start the `mybuild.xml` file by defining a project named `physiciantutorial`:

   ```
   <project name="tutorial" default=”build”>
   ```

4. Define the main target for building the application. This target (named “build”) is fairly simple. It uses the `wlcompile` task to identify the source directory (which uses the split development directory structure) and an output directory for storing compiled files. Enter the following lines:

   ```
     <target name="build">
       <wlcompile srcdir="c:/medrec_tutorial/src/physicianEar"
   destdir="c:/medrec_tutorial/build/physicianEar"/>
     </target>
   ```

   For most simple Enterprise Applications, you need only to point `wlcompile` to the source and build directories to use for compiling. Always make sure the `srcdir` and `destdir` directories point to separate locations—you want to ensure that your source and output files remain separate during the development process.

5. To complete the `mybuild.xml` file, add the following line to close the project:

   ```
   </project>
   ```

   Your completed file should resemble the following. Remember that you can copy over `wlcompile_tutorial.xml` if you do not want to type in the full text:

   ```
   <project name="tutorial" default="build">

     <target name="build">
       <wlcompile srcdir="c:/medrec_tutorial/src/physicianEar"
   destdir="c:/medrec_tutorial/build/physicianEar"/>
     </target>
   ```

```
</project>
```

Step 2: Compile the application.

After you create the `mybuild.xml` file, you can use it to compile the application.

1. Make sure you have set your environment using the MedRecDomain environment script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
   ```

2. Move to the `physicianEar` directory and compile by running the `mybuild.xml` script with ant:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ```

   ```
   ant -f mybuild.xml
   ```

   Although you did not add any informational messages to your build script, the `wlcompile` task produces its own output to show its progress:

   ```
   Buildfile: mybuild.xml

   mybuild:
     [javadoc] Generating Javadoc
     [javadoc] Javadoc execution
     [javadoc] Loading source file
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\m
   edrec\controller\PhysicianSessionEJB.java...
     [javadoc] Constructing Javadoc information...
     [javadoc] EJBGen 2.13beta
     [javadoc]     Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\m
   edrec\controller\PhysicianSessionHome.java
     [javadoc]     Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\m
   edrec\controller\PhysicianSession.java
     [javadoc]     Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\ejb-jar.x
   ml
     [javadoc]     Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\weblogic-
   ejb-jar.xml
     [javadoc]     Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\ejbgen-bu
   ild.xml
        [move] Moving 2 files to
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF
   ```

```
    [javac] Compiling 3 source files to
C:\medrec_tutorial\build\physicianEar\physSessionEjbs
    [javac] Compiling 13 source files to
C:\medrec_tutorial\build\physicianEar\physicianWebApp\WEB-INF\c
lasses

BUILD SUCCESSFUL

Total time: 4 seconds
```

3. If you did not receive the above output, you probably made a typo while creating the `mybuild.xml` file. If so, run the alternate compile command using the installed tutorial build file:

   ```
   ant -f wlcompile_tutorial.xml
   ```

Step 3: Examine the output files.

Now that you have compiled `physicianEar`, take a look at the build directory to see what happened. All output for the build target is placed in the output directory for the Enterprise Application, `c:\medrec_tutorial\build\physicianEar`.

The `wlcompile` output shows that the build started by running `ejbgen` on the Physician application's EJBs. Verify that the deployment descriptors were created:

```
dir c:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF

 Directory of
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF

02/21/2003  05:32p      <DIR>          .
02/21/2003  05:32p      <DIR>          ..
02/21/2003  05:32p                 697 ejb-jar.xml
02/21/2003  05:32p                 884 weblogic-ejb-jar.xml
```

`wlcompile` also compiled the and copied the actual EJB classes to the `physSessionEjbs` directory:

```
dir
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\med
rec\controller

 Directory of
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\med
rec\controller
```

```
02/21/2003  05:32p      <DIR>          .
02/21/2003  05:32p      <DIR>          ..
02/21/2003  05:32p              11,359 PhysicianClientUtils.class
02/21/2003  05:32p                 681 PhysicianSession.class
02/21/2003  05:32p               2,212 PhysicianSession.java
02/21/2003  05:32p               5,770 PhysicianSessionEJB.class
02/21/2003  05:32p               8,710 PhysicianSessionEJB.java
02/21/2003  05:32p                 324 PhysicianSessionHome.class
02/21/2003  05:32p                 428 PhysicianSessionHome.java
               7 File(s)         29,484 bytes
```

`wlcompile` compiled the Web Application servlet classes and placed them in the `WEB-INF\classes directory`:

```
dir
c:\medrec_tutorial\build\physicianEar\physicianWebApp\WEB-INF\cla
sses\com\bea\medrec

 Directory of
c:\medrec_tutorial\build\physicianEar\physicianWebApp\WEB-INF\cla
sses\com\bea\medrec

02/24/2003  10:53a      <DIR>          .
02/24/2003  10:53a      <DIR>          ..
02/24/2003  10:53a      <DIR>          actions
02/24/2003  10:53a      <DIR>          utils
```

The `actions` directory stores struts action classes and the `utils` directory contains a utility class that stores MedRec constants.

Notice that the entire build directory for the Enterprise Application (`c:\medrec_tutorial\build\physicianEar`) contains deployment descriptor files only for the EJB components. This is because the EJB descriptors are generated using `ejbgen` tags. You can recreate the entire contents of the build directory, including the EJB deployment descriptors, by rerunning the build script.

The Enterprise Application and Web Application deployment descriptors are left in the source directory because they are created and edited manually, and cannot be easily replaced or rebuilt.

Best Practices

More complex Enterprise Applications may have compilation dependencies that are not automatically handled by the `wlcompile` task. However, you can use the `include` and `exclude` options to `wlcompile` to enforce your own dependencies. `include` and `exclude` accept the names of Enterprise Application modules—the names of subdirectories in the Enterprise Application source directory—to include or exclude them from the compile stage. See The Big Picture for an example.

The Big Picture

Although the MedRec Enterprise Applications use the WebLogic split development directory structure and `wlcompile` task in their build scripts, they have certain dependencies that are not handled by the default `wlcompile` task. For example, examine this excerpt from the `medrecEar\build.xml` file:

```
<wlcompile srcdir="${medrec.ear.src.dir}" destdir="${dest.dir}"
         excludes="adminWebApp, xml, mdbEjbs, webServicesEjb"/>
```

You can see that the build script starts by compiling all modules in the Enterprise Application except for `adminWebApp`, `xml`, `mdbEjbs`, and `webServicesEjb`. These correspond to subdirectories names in the `medrecEar` source directory.

The build then continues by compiling *only* the `xml` and `webServicesEjb` modules in the application:

```
<wlcompile srcdir="${medrec.ear.src.dir}" destdir="${dest.dir}"
           includes="xml, webServicesEjb"
```

Related Reading

- Developing WebLogic Server Applications
- Developing Web Applications for WebLogic Server
- Programming WebLogic Server Enterprise JavaBeans
- Programming WebLogic Web Services

1 Building the MedRec Applications

Tutorial 8: Walkthrough of Web Application Deployment Descriptors

This tutorial examines the deployment descriptor files that define the resources and operating attributes of the MedRec Web applications.

Like most WebLogic Server Web Applications, each MedRec Web application uses two deployment descriptor files, `web.xml` and `weblogic.xml`. These files reside in the `WEB-INF` folders that are part of the directory structure of WebLogic Server Web Applications.

A `web.xml` deployment descriptor file is a J2EE standard XML document that sets properties for a Web Application. These properties are defined by the DTD referenced in a heading in each `web.xml` file, at `http://java.sun.com/dtd/web-app_2_3.dtd`.

A `weblogic.xml` deployment descriptor file is an XML document that defines WebLogic Server-specific properties for Web applications. These properties are defined by the DTD at `http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd`.

The tutorial includes the following sections:

- Prerequisites
- Procedure

Prerequisites

Before starting this tutorial:

- Create the MedRec domain and MedRec server. See Tutorial 1: Creating a WebLogic Domain and Server Instance for Development.
- Create the MedRec project directory. See Tutorial 5: Creating the MedRec Project Directory.
- Read about MedRec's split directory structure. See Tutorial 7: Compiling Applications Using the Split Development Directory.

Procedure

The following procedure walks you through the contents of the `web.xml` and `weblogic.xml` files.

Step 1: Examine a web.xml file.

In this section, examine how the `web.xml` file from `mainWebApp` configures `mainWebApp`'s resources. `mainWebApp` responds to HTTP requests in MedRec, either creating HTTP responses or forwarding requests to other Web components.

`web.xml` can define following attributes for a Web Application:

- Register servlets
- Define servlet initialization attributes

- Register JSP tag libraries
- Define security constraints
- Define other Web Application attributes

1. In a text editor, open the `web.xml` file that configures `mainWebApp`, located at `C:\medrec_tutorial\src\medrecEar\mainWebApp\WEB-INF`.
2. Note the required elements in the heading of the file, which set the encoding and point to the DTD that defines the elements and attributes that can be set in a `web.xml` file:

   ```
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <!DOCTYPE web-app
     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
   2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">
   ```

   Use caution with this section of the file, as erroneous edits here will likely prevent the application from being deployed until they are corrected.
3. The elements described in the following steps reside within the `web-app` element that they modify.

   ```
   <web-app>
   ....
   </web-app>
   ```
4. Note the registration of servlets in `web.xml`. The `servlet` element and its `servlet-class` attributes set the name of the servlet and the location of the compiled class that executes the servlet.

   The following listing names a servlet called "action," and associates it with a class:

   ```
   <servlet>
     <servlet-name>action</servlet-name>
     <servlet-class>org.apache.struts.action.ActionServlet</servle
   t-class>
   ```
5. `The init-param` attribute is part of the `servlet` element; in this case, of the servlet defined in the previous step. The servlet reads its `init-param` values when it is invoked.

```
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

6. The `servlet-mapping` element determines how the MedRec application invokes a servlet.

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

7. The `welcome-file-list` element defines the Web application's welcome files.

```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

8. `taglib` defines the tag libraries that are available to the application:

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

See web.xml Deployment Descriptor Elements in *Developing Web Applications for WebLogic Server*.

Step 2: Examine a weblogic.xml File

In this section, examine the contents of the `weblogic.xml` file that configures the `physicianWebApp`. Physicians and nurses log in to the physician Web Application to search and access patient profiles, create and review patient medical records, and prescribe medicine to patients.

A WebLogic Server Web Application's `weblogic.xml` file can set, among other things, the following major properties:

- JSP properties

- JNDI mappings
- Context root
- URL mappings
- Security role mappings
- HTTP session attributes

1. In a text editor, open the `weblogic.xml` file that configures `physicianWebApp`, located at `C:\medrec_tutorial\src\physicianEar\physicianWebApp\WEB-INF`.
2. Note the heading that references the DTD file:

   ```
   <!DOCTYPE weblogic-web-app
    PUBLIC "-//BEA Systems, Inc.//DTD Web Application 8.1//EN"
   "http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd"
   >
   ```

   This URL is the location of the current WebLogic Server 8.1 DTD for Web Applications.
3. The elements and attributes in the `weblogic.xml` file are members of the `weblogic-web-app` element that opens and closes every instance of `weblogic.xml`:

   ```
   <weblogic-web-app>
   ....
   </weblogic-web-app>
   ```
4. The `session-descriptor` element contains session parameters for the Web Application's servlet sessions. The parameter names refer to parameters specified in `weblogic810-web-jar.dtd`, whose values can be set within the same `session-descriptor` element.

   For example, the `InvalidationIntervalSecs` parameter is a performance-related setting that specifies the number of seconds the server waits before checking to determine if a session is invalid or has timed out.

   The next parameter, `TimeoutSecs`, sets the number of seconds the server waits before timing out a session.

The value assigned to the third parameter, `PersistentStoreType`, determines the persistent store method for servlet sessions. The current value, `replicated_if_clustered`, means that sessions on this server are stored in accordance with the value set for the cluster of servers to which this server belongs—if the Web Application is deployed to a cluster. Absent a clustered server configuration, servlet sessions default to the `memory PersistentStoreType`, in which sessions are not stored persistently.

```
<session-descriptor>
    <session-param>
      <param-name>InvalidationIntervalSecs</param-name>
      <param-value>60</param-value>
    </session-param>
    <session-param>
      <param-name>TimeoutSecs</param-name>
      <param-value>600</param-value>
    </session-param>
    <session-param>
      <param-name>PersistentStoreType</param-name>
      <param-value>replicated_if_clustered</param-value>
    </session-param>
</session-descriptor>
```

5. The `virtual-directory-mapping` element sets the location that the servlet checks first when fulfilling HTTP image requests. Its paired elements, `local-path` and `url-pattern`, map the URL pattern of an incoming request to a physical location.

```
<virtual-directory-mapping>
   <local-path>C:/bea/weblogic81/samples/server/medrec/src/comm
   on/web</local-path>
   <url-pattern>images/*</url-pattern>
   </virtual-directory-mapping>
```

6. The `context-root` element in a `weblogic.xml` file sets the context root directory for a Web Application. The context root is the base path of a Web application relative to the server's base URL. For example, MedRecServer's base

URL is `http://localhost:7101` and the Web application's context root is `physician`. Users access components of the `physician` Web application relative to `http://localhost:7101/physician`.

The setting `physician` means that users access the physicianWebApp when they specifically request it.

```
<context-root>physician</context-root>
```

See weblogic.xml Deployment Descriptor Elements in *Developing Web Applications for WebLogic Server*.

Best Practices

- Use an XML editor to edit XML files, rather than a text editor. It is easy to mishandle XML code, and you will save time by using an editor that validates your work.
- Use WebLogic Server tools to generate and edit XML deployment descriptors. WebLogic Builder generates descriptors and includes an interface for editing them—see WebLogic Builder Online Help at http://edocs.bea.com/wls/docs81/wlbuilder/index.html. DDInit generates descriptors for JARs, WARs, and EARs—see DDInit at http://edocs.bea.com/wls/docs81/admin_ref/utils.html#1170077.

The Big Picture

The MedRec application contains five Web Applications:

- physicianWebApp
- patientWebApp
- adminWebApp
- mainWebApp
- startupWebApp

The resources and attributes of these Web Applications are defined by deployment descriptor files. This tutorial describes the function of these deployment descriptors, specifically `web.xml`, the standard J2EE Web application deployment descriptor file, and `weblogic.xml`, the WebLogic Server-specific Web application deployment descriptor file.

Deployment descriptor files configure properties for MedRec's applications and EJBs, as well as its Web applications.

For example, `physicianEar`, the application to which `physicianWebApp` belongs, also contains a session EJB component, `physSessionEJBs`. `physSessionEJBs`'s deployment descriptor files, located at `C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF`, are the standard J2EE EJB deployment descriptor file `ejb-jar.xml`, and the WebLogic Server-specific EJB deployment descriptor file, `weblogic-ejb-jar.xml`.

`medrecEar`, the main MedRec application, is configured by a standard J2EE application deployment descriptor file, `application.xml`, located at `C:\medrec_tutorial\src\medrecEar\META-INF`.

You are encouraged to examine the EJB and application deployment descriptor files and the DTD files that they reference.

Related Reading

- Developing Web Applications for WebLogic Server, especially `web.xml` Deployment Descriptor Elements and `weblogic.xml` Deployment Descriptor Elements
- Sun's DTD for `web.xml`
- BEA WebLogic's DTD for weblogic.xml

1 Building the MedRec Applications

Tutorial 9: Deploying MedRec from the Development Environment

This tutorial describes how to deploy an application from a WebLogic split development directory using the `wldeploy` ant task and `weblogic.Deployer` utilities. You can use these techniques to deploy an application quickly to a development environment without having to package the application or otherwise modify your build environment.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial, complete tutorials 5 through 8 to create the project directory and perform the intermediate build steps for the Physician Application. If you completed tutorial 5 but skipped one or more of the subsequent tutorials, you can catch up by moving to the `c:\medrec_tutorial\src\physicianEar` subdirectory, setting the environment, and executing the Ant command:

```
ant -f build.xml
```

Procedure

By now you have seen how the split development directory structure helps you easily build Enterprise Applications with WebLogic Server. But deploying Enterprise Applications sometimes seems like as much work as building them—you usually need to combine the compiled Java classes with modifiable deployment descriptors to create an exploded EAR directory or a compressed EAR file, which you then deploy. This process generally involves copying files from one place to another and changing their directory structures before deploying (not to mention repeating this process each time you rebuild the application or change a deployment descriptor).

With the split development directory, compiled files in the `build` directory are neatly separated from modifiable source files and descriptors in the `source` directory. WebLogic Server can deploy applications directly from a split development directory—you only need to target the build directory to deploy your work. In this procedure you use the split development directory to deploy `physicianEar`, which has now been built to the point where it is deployable:

1. First start the `MedRecServer` if it is not already running:

   a. Open a new command shell:

      ```
      start cmd
      ```

   b. Start the MedRec server by running its start script:

      ```
      c:\bea\user_projects\domains\medrecdomain\startweblogic.cmd
      ```

2. Open a command shell and start PointBase, if it is not already running:

   ```
   cd c:\bea\weblogic81\common\eval\pointbase\tools
   ```

```
startPointBase.cmd
```

3. Open another command shell and set your environment:

```
c:\bea\user_projects\domains\medrecdomain\setenv.cmd
```

4. Move to the `physicianEar` subdirectory if you are not already there:

```
cd c:\medrec_tutorial\src\physicianEar
```

5. Use a text editor to create a new file, `deploy.xml`:

```
notepad deploy.xml
```

   Note: If you do not want to create the `deploy.xml` file manually in this tutorial, copy the file named `wldeploy_tutorial.xml` to a new file named `deploy.xml` and follow along.

6. Start the `deploy.xml` file by defining a project named `physiciandeploy`:

```
<project name="tutorial" default="deploy">
```

7. Define the main target for deploying the application:

```
  <target name="deploy">
     <wldeploy user="weblogic" password="weblogic"
adminurl="t3://127.0.0.1:7101" action="deploy"
name="tutorial_deployment"
source="c:\medrec_tutorial\build\physicianEar"/>
  </target>
```

8. Complete the `deploy.xml` file by closing the project element:

```
</project>
```

9. Your file contents should now resemble the following:

```
<project name="tutorial" default="deploy">

   <target name="deploy">
      <wldeploy user="weblogic" password="weblogic"
adminurl="t3://127.0.0.1:7101" action="deploy"
name="tutorial_deployment"
source="c:\medrec_tutorial\build\physicianEar" />
   </target>

</project>
```

   Save the file and exit your text editor.

10. In the same command shell, enter the commands to execute the build script:

```
ant -f deploy.xml
```

You should receive the following output from the `wldeploy` task:

```
Buildfile: deploy.xml

deploy:
 [wldeploy] weblogic.Deployer -noexit -name tutorial_deployment
-source C:\medrec_tutorial\build\physicianEar -adminurl
t3://127.0.0.1:7101 -user weblogic -password weblogic -deploy
 [wldeploy] Initiated Task: [0] [Deployer:149026]Deploy
application tutorial_deployment on MedRecServer.
 [wldeploy] Task 0 completed: [Deployer:149026]Deploy
application tutorial_deployment on MedRecServer.
 [wldeploy] Deployment completed on Server MedRecServer
 [wldeploy]

BUILD SUCCESSFUL

Total time: 12 seconds
```

If you do not receive the above output, MedRecServer may not have finished starting up, or you may have made a typo in creating the `deploy.xml` file. If this occurs, wait until the server has finished starting up, and try to deploy using the installed tutorial file:

```
ant -f wldeploy_tutorial.xml
```

11. To verify that the application deployed, open a new browser window and enter the URL, `http://127.0.0.1:7101/physician`. You should receive the Physician Application's login page. You cannot do much more than look at the page right now, because the rest of the MedRec application suite is not available.

12. The `wldeploy` task works using the same options as those available with the `weblogic.Deployer` command line utility. Before moving on to the next tutorial, undeploy the Physician application using `weblogic.Deployer`. In the same command-line window, enter the command:

    ```
    java weblogic.Deployer -adminurl t3://127.0.0.1:7101 -user
    weblogic -password weblogic -undeploy -name tutorial_deployment
    ```

    The utility displays the following output messages:

    ```
    Initiated Task: [1] [Deployer:149026]Remove application
    tutorial_deployment on MedRecServer.
    Task 1 completed: [Deployer:149026]Remove application
    tutorial_deployment on MedRecServer.
    Deployment completed on Server MedRecServer
    ```

Best Practices

- You can use the either the `weblogic.Deployer` tool or its associated Ant task, `wldeploy` to target the build directory to a server running in development mode. You cannot use the Administration Console to deploy from a split development directory.
- The split development directory structure enables you to deploy applications directly from your development environment without packaging or otherwise copying any files.
- In most cases, you need to deploy and redeploy frequently during the development phase of an Enterprise Application. You should generally add deploy and redeploy targets to your build files to your project build scripts to facilitate these functions. To redeploy using the `wldeploy` task, simply replace `action="deploy"` with `action="redeploy"`, and omit the `source` definition; `wldeploy` uses the deployment name to redeploy the application.

The Big Picture

How does `wldeploy` work with the split directory? The contents of `c:\medrec_tutorial\build\physicianEar` look similar to an exploded EAR directory, but there are no deployment descriptors. WebLogic Server finds the correct deployment descriptors to use by examining the `c:\medrec_tutorial\build\physicianEar\.beabuild.txt` file, which references the application's source directory, `c:\medrec_tutorial\src\physicianEar`. The source directory contains the component deployment descriptors needed to deploy the application.

Related Reading

- Performing Common Deployment Tasks
- Deployment Tools Reference

1 Generating Deployment Descriptors

Tutorial 10: Using EJBGen to Generate EJB Deployment Descriptors

This tutorial demonstrates how to use the WebLogic Server EJBGen utility to generate deployment descriptor files and EJB source files such as the home interface file.

The demonstration uses the PhysicianSession EJB from the Physician application in MedRec. You use EJBGen to generate new EJB source files and new versions of the EJB deployment descriptor files for the Physician EJB.

You compare the original versions of the deployment descriptor files to the newly generated versions. The files are:

- `ejb-jar.xml`, which specifies `PhysicianSessionEJB`'s bean, its interfaces, and its session and transaction types
- `weblogic-ejb-jar.xml`, which specifies `PhysicianSessionEJB`'s pool and time-out deployment settings

EJBGen uses annotations in the bean file to generate the deployment descriptor files and the EJB Java source files. EJB files in the MedRec application are already annotated for EJBGen.

The tutorial includes the following sections:

- Prerequisites
- Procedures
- Best Practices
- The Big Picture
- Related Reading

Prerequisites

Before starting this tutorial, complete Tutorial 9: Deploying MedRec from the Development Environment. In this tutorial, it is assumed that MedRec is bundled as the `tutorial_deployment` application and is ready to deploy to the MedRec server, as it is at the end of Tutorial 9.

Procedures

In the following procedures you view some of the files that EJBGen generates, use EJBGen to regenerate those files, redeploy the application, and then view the newly generated files.

- Procedure 1: Deploy the application and view the deployment descriptor files.
- Procedure 2: Generate new deployment descriptor and EJB files.
- Procedure 3: Redeploy the application and view the generated files.

Procedure 1: Deploy the application and view the deployment descriptor files.

1. Set your environment by opening a command window and running `setenv.cmd`:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setenv.cmd
   ```

2. Move to the `physicianEar` subdirectory:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ```

3. Redeploy the tutorial_deployment application. For example:

   ```
   ant -f deploy.xml
   ```

4. Open the Administration Console by navigating in a browser to `http://localhost:7101/console`.

5. In the left-hand panel of the Administration Console, expand Deployments-->Applications-->tutorial_deployments and click `physSessionEjb`.

6. In the right-hand panel, select the Descriptors tab.

7. Click the `ejb-jar.xml` file to view its text, so that you can compare it with the text you will use EJBGen to generate. The XML code quoted in this step and the next step is generated by EJBGen. You do not need to write it.

   The XML should appear as follows:

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>PhysicianSessionEJB</ejb-name>
      <home>com.bea.medrec.controller.PhysicianSessionHome</home>
      <remote>com.bea.medrec.controller.PhysicianSession</remote>

<ejb-class>com.bea.medrec.controller.PhysicianSessionEJB</ejb-cla
ss>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

8. Click the `weblogic-ejb-jar.xml` file. It should read as follows:

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
```

```
    <ejb-name>PhysicianSessionEJB</ejb-name>
    <stateless-session-descriptor>
      <pool>
        <max-beans-in-free-pool>1000</max-beans-in-free-pool>
        <initial-beans-in-free-pool>0</initial-beans-in-free-pool>
      </pool>
      <stateless-clustering>
      </stateless-clustering>
    </stateless-session-descriptor>
    <transaction-descriptor>
      <trans-timeout-seconds>0</trans-timeout-seconds>
    </transaction-descriptor>
    <enable-call-by-reference>True</enable-call-by-reference>

<jndi-name>PhysicianSessionEJB.PhysicianSessionHome</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

Procedure 2: Generate new deployment descriptor and EJB files.

1. In the command window, change to the `C:\medrec_tutorial\src\physicianEar\physSessionEjbs\com\bea\medrec\controller` directory.
2. Change the suffix of `PhysicianSessionEJB.ejb` to `.java`. For example:

   ```
   copy PhysicianSessionEJB.ejb PhysicianSessionEJB.java
   ```

   The `.ejb` suffix exists to tell the `wlcompile` ant script to run EJBGen on the EJB source files when it first compiles MedRec as described in Tutorial 7: Compiling Applications Using the Split Development Directory.
3. Enter the following command to invoke EJBGen on `physicianSessionEJB`.

a. If you are using WebLogic Server 8.1 Service Pack 1 or higher, issue the following command:

```
java weblogic.tools.ejbgen.EJBGen PhysicianSessionEJB.java -d
C:\medrec_tutorial\build\physicianEar\physSessionEjbs
-descriptorDir
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF
```

The -d flag tells EJBGen to write the Java files to the `C:\medrec_tutorial\build\physicianEar\physSessionEjbs` directory. The `-descriptorDir` flag specifies the directory for the deployment descriptor files relative to the output directory specified with the `-d` flag.

EJBGen reports on its progress as follows:

```
Loading source file PhysicianSessionEJB.java...

Constructing Javadoc information...

EJBGen 2.15

    Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\m
edrec\controller\PhysicianSessionHome.java

    Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\m
edrec\controller\PhysicianSession.java

    Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
ejb-jar.xml

    Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
weblogic-ejb-jar.xml

    Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
ejbgen-build.xml
```

b. If you are using a WebLogic Server 8.1 release earlier than Service Pack 1, your version of EJBGen does not support the `-descriptorDir` flag, so you will have to create the `C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\`directory and manually copy the generated files to it. Begin by issuing the following command:

```
java weblogic.tools.ejbgen.EJBGen PhysicianSessionEJB.java
```

EJBGen reports on its output:

```
[Info:]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\\ejb-jar.
xml

[Info:]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\\weblogic
-ejb-jar.xml

[Info:]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\ejbgen-bu
ild.xml13 warnings
```

Next, move the newly generated descriptor files to their proper directories as listed below:

```
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
ejb-jar.xml

C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
weblogic-ejb-jar.xml

C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\
ejbgen-build.xml
```

Procedure 3: Redeploy the application and view the generated files.

1. Move to the `physicianEar` subdirectory:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ```

2. Redeploy the tutorial_deployment application. For example:

   ```
   ant -f deploy.xml
   ```

3. Open the Administration Console by navigating in a browser to `http://localhost:7101/console`.

4. In the left-hand panel of the Administration Console, expand Deployments>Applications>tutorial_deployment.

5. Click `physSessionEjbs`. The right pane shows the configuration of the EJB module.

6. In the right pane, select the Descriptors tab.

7. Click `ejb-jar.xml` and `weblogic-ejb-jar.xml` to view them and compare them to the original versions you viewed in the first procedure.

   The new versions of `PhysicianSessionHome.java` and `PhysicianSessionHome.java` are in the

`C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\controller` directory.

Best Practices

Use EJBGen to develop the EJB component your application. You can simplify your EJB development and code maintenance by writing just the bean files and annotating them with EJBGen tags, and then generating all the remaining files—the home interface, the local interface, the deployment descriptor files—using EJBGen.

The Big Picture

The scripts that compile and deploy MedRec use EJBGen to generate most of the EJB files in the application.

The PhysicianSession bean contains all of the information necessary for EJBGen to generate the EJB descriptor files and the home interface. You can view the EJBGen annotations by opening `C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF\com\bea\medrec\controller>PhysicianSession.java` in a text editor.

For example, the following tags define the pool and timeout settings that you see in the generated `weblogic-ejb-jar.xml`:

```
/**
 * <p>Session Bean implementation for physician functionality
including
 * access MedRec web services.</p>
 *
 * @author Copyright (c) 2003 by BEA Systems. All Rights Reserved.
 *
 * EJBGen tags:
 * @ejbgen:session
 *   max-beans-in-free-pool = 1000
```

```
*   initial-beans-in-free-pool = 0
*   trans-timeout-seconds = 0
*   type = Stateless
*   enable-call-by-reference = True
*   ejb-name = PhysicianSessionEJB
*
* @ejbgen:jndi-name
*   remote = PhysicianSessionEJB.PhysicianSessionHome
*/
```

Related Reading

- EJBGen Reference
- WebLogic Server Workshop

  In the left-hand navigational area, expand the Developing Enterprise Java Beans topic to learn about using WebLogic Server Workshop to create EJBs.
- Deployment Descriptors, in *Programming WebLogic Enterprise JavaBeans*

1 Building the MedRec Applications

Tutorial 11: Exposing a Stateless Session EJB as a Web Service

This tutorial describes how to expose a stateless session EJB as a Web Service. Tutorial 12: Invoking a Web Service from a Client Application describes how this Web Service can be invoked by a variety of client applications using SOAP.

WebLogic Web Services Ant tasks expose an existing stateless session EJB as a WebLogic Web Service. In particular, the Ant tasks:

- Generate a `web-services.xml` deployment descriptor file that tells WebLogic Server how to deploy the Web Service.
- Generate the serialization classes, XML Schema representation, and type mapping information for all non-built-in data types that are used as parameters and return values of the EJB methods.
- Package all the components into a Web Application within an EAR file, along with any needed EJB JAR files.

This tutorial shows you how to use the individual Ant tasks `autotype` and `source2wsdd`. BEA also provides another Ant task, `servicegen`, which can automate many tasks for you.

The tutorial includes the following sections:

Prerequisites

Previous tutorials work exclusively with the Physician application, `physicianEar`. This tutorial uses the main MedRec application, `medrecEar`, which contains the session and entity EJBs that handle patient information such as personal data and records of doctor visits.

It is assumed that you have already created the split directory structure for the MedRec application, and compiled the EJBs that make up the application (in particular the `MedRecWebServices` stateless session EJB) into their respective class files. To accomplish these tasks, you must:

1. Create the project directory and copy over the source files and output directories using the instructions in Tutorial 5: Creating the MedRec Project Directory.
2. Change to the `medrecEar` subdirectory in the MedRec project directory:

   ```
   cd c:\medrec_tutorial\src\medrecEar
   ```

3. Set your environment using the `MedRecDomain` environment script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
   ```

4. Execute the existing `build.xml` Ant file, specifying the `prepare` and `build.split.dir` targets:

   ```
   ant prepare build.split.dir
   ```

   The `prepare` target creates needed directories, sets up the Web Application directories, and copies the shared utility JAR files to the build directory. The `build.split.dir` target compiles all the EJBs into class files.

   After running the Ant task, examine the `c:\medrec_tutorial\build\medrecEar` directory; you will see subdirectories representing all the EJBs of the MedRec application, as well as

Web Application directories. The `ws_medrec` Web Application will contain the Web Services information. This tutorial shows how to convert the stateless session EJB located in the `webServicesEJB` directory, `com.bea.medrec.webservices.MedRecWebServices`, into a WebLogic Web Service.

Procedure

To expose the `MedRecWebServices` EJB as a Web Service, follow these steps:

- Step 1: Create the build file that contains calls to the Web Services Ant tasks.
- Step 2: Execute the Web Services Ant tasks and create the Web Service.
- Step 3: Deploy the Web Service and view its home page.

Step 1: Create the build file that contains calls to the Web Services Ant tasks.

WebLogic Server provides a variety of Ant tasks that help you expose an existing stateless session EJB as a Web Service. This tutorial shows how to use the following Ant tasks:

- `autotype`: Generates the serialization classes, XML Schema representation, and type mapping information for the non-built-in Java data types that are used as parameters and return values of the EJB methods. The serialization class converts data between its XML and Java representations during the invoke of a Web Service operation.
- `source2wsdd`: Generates the `web-services.xml` deployment descriptor file that describes the Web Service. The `source2wsdd` Ant task uses information generated from the `autotype` task (such as the type mapping information) as well as information from the EJB being exposed (such as its methods and parameters) to generate the `web-services.xml` file.

For detailed information about these Ant tasks, see *Web Service Ant Tasks and Command-Line Utilities*.

In this tutorial, all components will be generated directly into the `ws_medrec` Web Application of the MedRec build directory.

1. Change to the `medrecEar` subdirectory in the MedRec project directory:

```
cd c:\medrec_tutorial\src\medrecEar
```

2. Use your favorite text editor to create a file called `my_webserv.xml` file (which will contain calls to the Ant tasks) in the `medrecEar` directory:

   ```
   notepad my_webserv.xml
   ```

   Note: If you do not want to enter the build file manually, copy the file `webservices_tutorial.xml` file to the new file name, `my_webserv.xml`. Then follow along to understand the file contents. The `webservices_tutorial.xml` file assumes that your MedRec project directory is `c:/medrec_tutorial`.

3. Add the following lines to the `my_webserv.xml` file (substituting, if necessary, your actual MedRec project directory for `c:/medrec_tutorial`):

```
<project name="WebServicesTutorial" default="build.ws">

 <target name="build.ws">

   <autotype
        javaComponents="com.bea.medrec.webservices.MedRecWebServices"
        targetNamespace="http://localhost:7101/ws_medrec/MedRecWebServices"
        packageName="com.bea.medrec.webservices"
        earClasspath="c:/medrec_tutorial/build/medrecEar"
        keepGenerated="false"
        destDir="c:/medrec_tutorial/build/medrecEar/ws_medrec/WEB-INF/classes"
    />

    <source2wsdd

javaSource="webServicesEjb/com/bea/medrec/webservices/MedRecWebServices.java"
        ejbLink="webServicesEjb#MedRecWebServicesEJB"

typesInfo="c:/medrec_tutorial/build/medrecEar/ws_medrec/WEB-INF/classes/types.x
ml"

ddFile="c:/medrec_tutorial/build/medrecEar/ws_medrec/WEB-INF/web-services.xml"
        serviceURI="/MedRecWebServices"
        earClasspath="c:/medrec_tutorial/build/medrecEar"

classpath="${java.class.path};c:/medrec_tutorial/build/medrecEar/ws_medrec/WEB-
INF/classes"
        wsdlFile="c:/medrec_tutorial/dist/MedRecService.wsdl"
  />

  </target>

</project>
```

Table 1: Attributes of the autotype and source2wsdd Ant Tasks

Ant Task	Attribute	Description
autotype	javaComponents	The class name of the remote interface of the MedRecWebServices stateless session EJB. The autotype Ant task introspects this class to find the list of non-built-in Java data types for which it needs to create the serialization class, XML Schema representation, and type mapping information.
	targetNamespace	Namespace URI of the Web Service.
	packageName	Package name of the generated serialization classes.
	earClasspath	Specifies that the EJB JAR files and classes in the APP-INF directory of the MedRec application build directory be added to the CLASSPATH of the autotype Ant task. This ensures that autotype finds the compiled Java classes of the MedRecWebServices EJB, as well as the compiled classes of non-built-in data types, that were generated as a prerequisite to this tutorial.
	keepGenerated	Specifies that only the class files, and not the Java source files, of the generated serialization classes should be generated to the build directory.
	destDir	Full pathname of the directory that will contain the generated components. The serialization classes will be placed in a directory structure that mirrors their package name and the XML Schema representation and type mapping information is generated in a file called types.xml. In this tutorial, the components are generated directly into the WEB-INF directory of the ws_medrec Web Application of the MedRec enterprise application.

Ant Task	Attribute	Description
source2wsdd	javaSource	Name of the Java source file of the MedRecWebService EJB. In the tutorial, this is a relative path, starting with webServicesEjb, which contains the Java source files for the MedRecWebService EJB.
	ejbLink	At runtime, WebLogic Server uses this element to determine the name of the stateless session EJB, and the EJB JAR file in which it is contained, that is being exposed as a Web Service. In particular, this attribute specifies the value of the <ejb-link> child element of the <stateless-ejb> element in the generated web-services.xml file.
	typesInfo	Full pathname of the types.xml file that contains the XML Schema representation and type mapping information for the non-built-in data types. This file was generated by the autotype Ant task.
	ddFile	Full pathname of the generated web-services.xml deployment descriptor file. In this tutorial, the file is generated directly into the WEB-INF directory of the ws_medrec Web Application of the MedRec application, medrecEar.
	serviceURI	The Web Service URI portion of the URL used by client applications to invoke the deployed Web Service.
	earClasspath	Specifies that the EJB JAR files and classes in the APP-INF directory of the MedRec application build directory be added to the CLASSPATH of the source2wsdd Ant task. This ensures that source2wsdd finds all the compiled classes that were generated as a prerequisite to this tutorial.
	classpath	Adds the classes in the WEB-INF/classes directory of the ws_medrec Web Application of the build directory to the CLASSPATH of the source2wsdd Ant task. The ws_medrec/WEB-INF/classes directory contains the serialization classes generated by the autotype Ant task.
	wsdlFile	Specifies that you also want to generate a hard copy of the WSDL (public contract of the Web Service) into the specified directory. **Note:** The generated WSDL is used in Tutorial 12: Invoking a Web Service from a Client Application to generate a client JAR file that contains much of the Java code needed by Java client applications to invoke the Web Service. This step is performed now *only* to set up a later tutorial; you typically never need to create a static copy of the WSDL file. This is because

Step 2: Execute the Web Services Ant tasks and create the Web Service.

After you have created the `my_webserv.xml` file, use it to execute the `autotype` and `source2wsdd` Ant tasks to create the Web Service components:

1. Set your environment using the `MedRecDomain` environment script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
   ```

2. Move to the `medrecEar` directory:

   ```
   cd c:\medrec_tutorial\src\medrecEar
   ```

3. Execute the Web Service Ant tasks by running the `my_webserv.xml` script using `ant`:

   ```
   ant -f my_webserv.xml
   ```

   Although you did not add any informational messages to your build script, the `autotype` and `source2wsdd` Ant tasks produce output to show their progress:

```
Buildfile: my_webserv.xml

build.ws:
 [autotype] Autotyping for javaComponents
com.bea.medrec.webservices.MedRecWebServices

[source2wsdd] Loading source file
c:\medrec_tutorial\src\medrecEar\webServicesEjb\com\bea\medrec\webservices\MedR
ecWebServices.java...

[source2wsdd] Constructing Javadoc information...

BUILD SUCCESSFUL

Total time: 17 seconds
```

4. Move to the MedRec build directory to see what the Ant tasks generated:

   ```
   cd c:\medrec_tutorial\build\medrecEar
   ```

   The generated Web Service components are in subdirectories of the `ws_medrec` Web Application directory:

   - The `web-services.xml` file is in the `WEB-INF` directory.
   - The serialization classes are in the `WEB-INF\classes` directory, in a sub-directory structure that corresponds to the package name.

Step 3: Deploy the Web Service and view its home page.

In this section, deploy the entire MedRec application, which includes the `MedRecWebService` Web Service, in the same way that you deployed the Physician application in Tutorial 9: Deploying MedRec from the Development Environment.

Once the Web Service is deployed, you can view its home page where you can test its operations, view the WSDL of the Service, and so on.

1. Start `MedRecServer`, if it is not already running, by executing its start script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\startweblogic.cmd
   ```

2. Open another command shell and set your environment:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setenv.cmd
   ```

3. Move to the `medrecEar` subdirectory if you are not already there:

   ```
   cd c:\medrec_tutorial\src\medrecEar
   ```

4. Use your favorite text editor to edit the `my_webserv.xml` file:

   ```
   notepad my_webserv.xml
   ```

5. Add a new target to the file by adding the following text directly after the `</target>` tag which ends the existing `build.ws` target:

   ```
     <target name="deploy">
        <wldeploy user="weblogic" password="weblogic"
   adminurl="t3://localhost:7101" action="deploy"
   name="medrec_deployment"
   source="c:\medrec_tutorial\build\medrecEar"/>
     </target>
   ```

   Note: If you do not want to enter the text manually, copy the text from the file `webservices_tutorial.xml` file. Then follow along to understand the file contents. The `webservices_tutorial.xml` file assumes that your MedRec project directory is `c:/medrec_tutorial`.

   Save the file and exit your text editor.

6. In the same command shell, enter the following command to execute just the `deploy` target of the build script:

   ```
   ant -f my_webserv.xml deploy
   ```

   You should receive the following output from the `wldeploy` task:

   ```
   Buildfile: my_webserv.xml
   ```

```
deploy:

[wldeploy] weblogic.Deployer -noexit -name medrec_deployment
-source D:\medrec_tutorial\build\medrecEar -adminurl
t3://localhost:7101 -user weblogic -password weblogic -deploy

 [wldeploy] Initiated Task: [3] [Deployer:149026]Deploy
application medrec_deployment on MedRecServer.

 [wldeploy] Task 3 completed: [Deployer:149026]Deploy
application medrec_deployment on MedRecServer.

 [wldeploy] Deployment completed on Server MedRecServer

 [wldeploy]

BUILD SUCCESSFUL

Total time: 18 seconds
```

If you do not receive the above output, MedRecServer may not have finished starting up, or you may have made a typo in creating the `deploy` target in the `my_webserv.xml` file. If this occurs, wait until the server has finished starting up, and try to deploy using the installed tutorial file (it is assumed that the MedRec project directory is `c:\medrec_tutorial`):

```
ant -f webservices_tutorial.xml deploy
```

7. To verify that the Web Service deployed, open a new browser window and enter the URL for the Web Service's home page:

   ```
   http://localhost:7101/ws_medrec/MedRecWebServices
   ```

   From the home page you can test the operations of the Web Service by clicking on the operation name; view the WSDL by clicking on the Service Description link; and view the SOAP request and response messages of a successful invocation of the operations.

Best Practices

- A WebLogic Web Service can be implemented with a stateless session EJB or a Java class. Use a stateless session EJB when you want to take advantage of standard EJB features, such as transactions, pooling, and so on. Use a Java class if you do not need these features and you want to develop or prototype a Web Service quickly.

- When creating a Web Service, use a stateless session EJB or Java class facade that exposes a simple interface to your application. This facade will not necessarily do much other than call other session EJBs, which in turn might call entity EJBs, that do the work. Encapsulating the main application logic into one EJB facade that you expose as a Web Service, rather than exposing all the session EJBs of your application, makes the public interface to your application simpler and cleaner. It also allows you to change the supporting EJBs without having to change the public face of your application.
- Think about the non-built-in XML data types that your application uses and whether they will interoperate with all client applications that will be invoking your Web service. For example, although the `autotype` Ant task can create a serialization class and XML Schema for the `java.utils.Collection` Java data type, the resulting XML Schema data type might not necessarily interoperate with all client applications.

  For this reason, consider using simpler data types (such as Arrays) in the EJB facade that will be exposed as a Web Service, and then, inside of the EJB facade, convert these data types into the more complex Java types (such as `java.utils.Collection`) used by the session EJBs that do the actual work.
- WebLogic Server provides a variety of Ant tasks to help you expose an EJB or Java class as a Web Service. If your Web Service is fairly simple and straightforward, use the `servicegen` Ant task which does everything for you, including optionally configuring your Web Service for security, reliable SOAP messaging, and handler chain, as well as packaging the Web Service into an EAR file. If, however, you want more control over the various stages of assembling the Web Service, and want to package it yourself as part of a bigger J2EE application, then use the individual Ant tasks that are targeted for specific jobs, such as `autotype` and `source2wsdd`. This tutorial shows how to use the individual Ant tasks.
- Every deployed WebLogic Web Service has a home page from which you can perform preliminary and simple testing of the service's operations, view its WSDL, and view SOAP messages of successful invokes of the operations. Use the home page for a first-pass testing of your Web Service; use a client application for more rigorous testing. See *Deploying and Testing WebLogic Web Services* for detailed information on the URL used to invoke the home page.

The Big Picture

The `com.bea.medrec.webservices.MedRecWebServices` stateless session EJB of the MedRec application contains methods to view and update patient and record information, such as `addRecord()`, `updatePatient()` and so on. These methods do not actually perform any of the business logic; rather, they call the existing session EJBs (such as `com.bea.medrec.controller.PatientSession` and `com.bea.medrec.controller.RecordSession`) to do the real work of viewing and searching for the patient and record information. You can think of the `MedRecWebServices` EJB as a facade that takes incoming requests to the MedRec application and hands them off to the other session and entity EJBs that do the actual work.

For this reason, the `MedRecWebServices` EJB is a good candidate to be exposed as a Web Service so that all kinds of different client applications, from EJBs running on a different WebLogic Server instance to a .NET client, can easily get to and update the patient and record information managed by the MedRec application. The client applications use SOAP to invoke a Web Service operation, and WebLogic Server in turn uses SOAP to send the information back to the client.

The methods of the `MedRecWebServices` EJB use the following complex non-built-in data types as parameters and return values:

- `AddressWS`
- `PatientWS`
- `PrescriptionWS`
- `RecordWS`
- `RecordSummaryWS`
- `VitalSignsWS`

These data types are almost exactly the same as the `com.bea.medrec.value.*` value objects used by the other session and entity EJBs of the MedRec application. The only difference is that the Web Service-specific ones do not use Java types such as `java.util.List` and `java.util.Collection` to represent collections of data, but use arrays. The reason for this is that arrays are much more interoperable and type-bound than `List` and `Collection`. The `autotype` Ant task creates the serialization class to convert between the Web Service data types and their equivalent

XML Schema type, and then the `MedRecWebServices` EJB converts the data between the Web Service Java data type (such as `AddressWS`) and its equivalent Value object (such as `Address`).

Related Reading

- *Programming WebLogic Web Services*
- *Programming WebLogic XML*
- *Programming WebLogic Enterprise Java Beans*
- Simple Object Access Protocol (SOAP) 1.1 Specification
- Web Services Description Language (WSDL) 1.1 Specification

1 Building the MedRec Applications

Tutorial 12: Invoking a Web Service from a Client Application

This tutorial describes how to invoke the `MedRecWebServices` WebLogic Web Service you created in Tutorial 11: Exposing a Stateless Session EJB as a Web Service from the following types of client applications:

- Stateless session EJB deployed to WebLogic Server
- Stand-alone Java Swing client application
- .NET client

Stateless session EJBs and stand-alone Java clients use the Web Service-specific client JAR file, generated by the `clientgen` Ant task, that contains most of the Java code you need to invoke a Web Service. The .NET client is written in C# and is provided to show that you can invoke a WebLogic Web Service from non-Java clients as well.

Note: You can use the `clientgen` Ant task to generate the JAX-RPC stubs for Web Services deployed on both WebLogic Server *and* other application servers.

The tutorial includes the following sections:

- Prerequisites
- Procedures

Prerequisites

It is assumed that you already know how to create a session EJB, a stand-alone Java Swing client application, and a .NET client application, and you want to learn how to update them to invoke a Web Service.

Before starting this tutorial, complete tutorials 5 through 11 to create the project directory and perform the intermediate build steps for the Physician and Medrec Applications. If you skipped any of the tutorials 5 through 11, you can catch up by following these steps:

1. Set your environment:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setenv.cmd
   ```

2. Move to the `c:\medrec_tutorial\src\physicianEar` subdirectory and execute the `ant` command:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ant -f wlcompile_tutorial.xml
   ant -f wldeploy_tutorial.xml
   ```

3. Move to the `c:\medrec_tutorial\src\medrecEar` subdirectory and execute the `ant` command:

   ```
   cd c:\medrec_tutorial\src\medrecEar
   ant prepare build.split.dir
   ant -f webservices_tutorial.xml
   ant -f webservices_tutorial.xml deploy
   ```

   Note: In the *XXX*`_tutorial.xml` files, it is assumed that your MedRec project directory is `c:\medrec_tutorial`. If your project directory is different, you must update the files to ensure that they work correctly.

Procedures

The following procedures show the steps and code excerpts needed to invoke a Web Service from different types of client applications.

- Procedure 1: Invoke a Web Service from an EJB deployed on WebLogic Server.
- Procedure 2: Invoke a Web Service from a stand-alone Java Swing client application.
- Procedure 3: Invoke a Web Service from a .NET client.

Procedure 1: Invoke a Web Service from an EJB deployed on WebLogic Server.

This procedure describes how to invoke a Web Service from the `PhysicianSessionEJB` of the Physician application. The procedure shows you how to run the `clientgen` Ant task to generate most of the needed Java code into a client JAR file, then walks you through the code in the `PhysicianSessionEJB` used to invoke the Web Service.

1. Change to the `physicianEar` subdirectory of the MedRec project directory:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ```

2. Use your favorite text editor to create a file called `my_webserv_client.xml` file:

   ```
   notepad my_webserv_client.xml
   ```

3. Add the following lines to the `my_webserv_client.xml` file (substituting, if necessary, your actual MedRec project directory for `c:/medrec_tutorial`).

   Note: If you do not want to create the build file manually, copy the contents of the `ws_ejb_client_tutorial.xml` file to the new file, `my_webserv_client.xml`. Then follow along to understand the file contents. In the `ws_ejb_client_tutorial.xml` file, it is assumed that your MedRec project directory is `c:/medrec_tutorial`.

```
<project name="EJB Web Service Invoke" default="build.ws.client">

  <target name="build.ws.client">
    <clientgen
      wsdl="http://localhost:7101/ws_medrec/MedRecWebServices?WSDL"
      packageName="com.bea.medrec.webservices"
      keepGenerated="false"
```

```
clientjar="c:/medrec_tutorial/build/physicianEar/APP-INF/lib/webServicesEjb_cli
ent.jar" />

  </target>

</project>
```

The Ant build file calls the `clientgen` Web Services Ant task which generates a client JAR file that contains most of the Java code (in particular, the JAX-RPC stubs) you need to invoke a Web Service. The `wsdl` attribute specifies that the `clientgen` Ant task should use the WSDL of the WebLogic Web Service you deployed in Tutorial 11: Exposing a Stateless Session EJB as a Web Service when generating the client JAR file. The JAR file, called `webServicesEjb_client.jar`, is created in the `APP-INF/lib` build directory of the Physician application, `physicianEar`.

Note: In the preceding Ant build file, it is assumed that the `MedRecWebServices` WebLogic Web Service is deployed and its WSDL is accessible. If you have not yet deployed the Web Service, you can point the `wsdl` attribute to a static WSDL file, distributed as part of the MedRec tutorial JAR file. The static file is distributed as a convenience; typically you point `clientgen` to a dynamically generated WSDL to create the client JAR file. To use the static WSDL file, update the `my_webserv_client.xml` as shown in bold:

```
<project name="EJB Web Service Invoke" default="build.ws.client">

  <target name="build.ws.client">
    <clientgen
      wsdl="c:/medrec_tutorial/dist/MedRecService.wsdl"
      packageName="com.bea.medrec.webservices"
      keepGenerated="false"

clientjar="c:/medrec_tutorial/build/physicianEar/APP-INF/lib/webServicesEjb_cli
ent.jar" />

  </target>

</project>
```

4. Ensure you have set your environment using the `MedRecDomain` environment script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
   ```

5. Execute the `clientgen` Ant task by running the `my_webserv_client.xml` script:

```
ant -f my_webserv_client.xml
```

The `clientgen` Ant task shows the following output:

```
Buildfile: my_webserv_client.xml

build.ws.client:

[clientgen] Generating client jar for
http://localhost:7101/ws_medrec/MedRecWebServices?WSDL ...

BUILD SUCCESSFUL

Total time: 15 seconds
```

The `clientgen` Ant task automatically generates the client JAR file into the `APP-INF/lib` directory of the `physicianEar` build directory, which means that the JAR file is automatically added to the EJB's CLASSPATH when the EJB is deployed in development mode to WebLogic Server.

When you package the Physician application for production, package the Web Services client JAR file the same as any other supporting JAR files inside of the EJB JAR file.

6. Update the `PhysicianSessionEJB` to invoke the Web Service.

   Note: This part of the tutorial simply walks you through the EJB code you would write; the `PhysicianSessionEJB.ejb` code in the MedRec tutorial JAR file already contains the code needed to invoke the `MedRecWebServices` Web Service.

   a. Change to the directory that contains the `PhysicianSessionEJB` Java code:

```
cd c:\medrec_tutorial\src\physicianEar\physSessionEjbs\com\bea\medrec\controller
```

   b. Open the `PhysicianSessionEJB.ejb` file in your favorite editor:

```
notepad PhysicianSessionEJB.ejb
```

   c. Search for the private method `getMedRecWebServicesPort()`. This method contains the Java code that creates a JAX-RPC stub of the Web Service:

```
wsdl_url = System.getProperty("phys.app.wsdl.url");
logger.debug("Wsdl url: "+wsdl_url);
MedRecWebServices service = new
       MedRecWebServices_Impl(wsdl_url);
port = service.getMedRecWebServicesPort();
```

   The URL of the WSDL of the deployed `MedRecWebServices` is passed to the EJB using the `phys.app.wsdl.url` system property that was set in the

MedRecServer startup script in the first tutorial, Tutorial 1: Creating a WebLogic Domain and Server Instance for Development. The value of the system property is the WSDL of the Web Service:

```
http://localhost:7101/ws_medrec/MedRecWebServices?WSDL
```

d. The public methods of `PhysicianSessionEJB` use this JAX-RPC stub to invoke Web Service operations.

For example, search for the public method `addRecord()`. It contains the following Java code that invokes the `addRecord` operation of the `MedRecWebServices` Web Service:

```
RecordWS recordWS = PhysicianClientUtils.toRecordWS(pRecord);
port.addRecord(recordWS);
```

The `PhysicianClientUtils.toRecordWS()` method is a utility that converts the standard `Record` Value object to a Web Service-specific `RecordWS` data type, to ensure interoperability. For details, see The Big Picture.

7. Compile and run `PhysicianSessionEJB` as usual.

For information about compiling, see Tutorial 7: Compiling Applications Using the Split Development Directory.

Procedure 2: Invoke a Web Service from a stand-alone Java Swing client application.

This procedure shows how to invoke a Web Service from a stand-alone Java Swing client application. The procedure first describes how to run the `clientgen` Ant task to generate most of the needed Java code into a client JAR file and then walks you through the client code you need to write. It is assumed that you know how to write, compile, and run a Java Swing client application.

A stand-alone client application must update its CLASSPATH to include the client JAR file generated by the `clientgen` Ant task, as well as the runtime Web Services JAR file *`WL_HOME`*`\server\lib\webserviceclient.jar`, where *`WL_HOME`* refers to the top-level directory of WebLogic Platform.

1. Change to the `clients` subdirectory of the MedRec project directory:

```
cd c:\medrec_tutorial\src\clients
```

2. Use your favorite text editor to create a file called `my_webserv_client.xml` file:

   ```
   notepad my_webserv_client.xml
   ```

3. Add the following lines to the `my_webserv_client.xml` file (substituting, if necessary, your actual MedRec project directory for `c:/medrec_tutorial`).

   Note: If you do not want to create the build file manually, copy the contents of the file `ws_standalone_client_tutorial.xml` file to the new file, `my_webserv_client.xml`. Then follow along to understand the file contents. It is assumed that your MedRec project directory is `c:/medrec_tutorial`.

```
<project name="Standalone Web Service Invoke" default="build.ws.client" >

  <target name="build.ws.client">
    <clientgen
      wsdl="http://localhost:7101/ws_medrec/MedRecWebServices?WSDL"
      packageName="com.bea.medrec.webservices"
      keepGenerated="false"
      clientjar="c:/medrec_tutorial/build/clients/webServicesEjb_client.jar" />
  </target>

</project>
```

The Ant build file calls the `clientgen` Web Services Ant task which generates a client JAR file that contains most of the Java code (in particular, the JAX-RPC stubs) you need to invoke a Web Service. The `wsdl` attribute specifies that the `clientgen` Ant task should use the WSDL of the WebLogic Web Service you deployed in Tutorial 11: Exposing a Stateless Session EJB as a Web Service when generating the client JAR file. The JAR file, called `webServicesEjb_client.jar`, is created in the `clients` build directory.

Note: In the preceding Ant build file, it is assumed that the `MedRecWebServices` WebLogic Web Service is deployed and its WSDL is accessible. If you have not yet deployed the Web Service, you can point the `wsdl` attribute to a static WSDL file, distributed as part of the MedRec tutorial JAR file. The static file is distributed as a convenience; typically you point `clientgen` to a dynamically generated WSDL to create the client JAR file. To use the static WSDL file, update the `my_webserv_client.xml` as shown in bold:

```
<project name="Standalone Web Service Invoke" default="build.ws.client" >

  <target name="build.ws.client">
    <clientgen
      wsdl="c:/medrec_tutorial/dist/MedRecService.wsdl"
```

```
      packageName="com.bea.medrec.webservices"
      keepGenerated="false"
      clientjar="c:/medrec_tutorial/build/clients/webServicesEjb_client.jar" />
  </target>

</project>
```

4. Ensure you have set your environment using the MedRecDomain environment script:

   ```
   c:\bea\user_projects\domains\MedRecDomain\setEnv.cmd
   ```

5. Execute the `clientgen` Ant task by running the `my_webserv_client.xml` script:

   ```
   ant -f my_webserv_client.xml
   ```

   The `clientgen` Ant task shows the following output:

```
Buildfile: my_webserv_client.xml

build.ws.client:

[clientgen] Generating client jar for
http://localhost:7101/ws_medrec/MedRecWebServices?WSDL ...

BUILD SUCCESSFUL

Total time: 14 seconds
```

6. Update the stand-alone Java Swing client application to invoke the Web Service.

   Note: This part of the tutorial simply walks you through the Java code you would write; the Java Swing client application of the MedRec tutorial JAR file already contains the code needed to invoke the `MedRecWebServices` Web Service.

   a. Change to the directory that contains the Java Swing client application code:

```
cd c:\medrec_tutorial\src\clients\com\bea\medrec\webservices\swing
```

   b. Open the `EditProfileFrame.java` file in your favorite editor:

      ```
      notepad EditProfileFrame.java
      ```

   c. Search for the method `submitButton_actionPerformed(ActionEvent e)` which returns patient information, based on the patient's social security number, when a user of the application clicks Submit. This method contains the following Java code:

```
MedRecWebServices ws = new
   MedRecWebServices_Impl(this.WSDLTextField.getText());
MedRecWebServicesPort port  = ws.getMedRecWebServicesPort();

PatientWS patientWS =
   (PatientWS)port.findPatientBySsn(this.patientIDTextField.getText());
```

The preceding code shows how to create a JAX-RPC stub of the `MedRecWebServices` Web Service from the WSDL in the `WSDLTextField` of the application, and then invoke the `findPatientBySsn` Web Service operation.

d. Search for the method `saveButton_actionPerformed(ActionEvent e)`, which saves updated patient information to the MedRec application by invoking the `updatePatient` Web Service operation:

```
PatientWS patientWS = Utils.toPatientWS(patient);
port.updatePatient(patientWS);
```

The `Utils.toPatientWS()` method is a utility that converts the standard `Patient` Value object to a Web Service-specific `PatientWS` data type, to ensure interoperability. For details, see The Big Picture.

7. Change to the main source directory for the client applications:

```
cd c:\medrec_tutorial\src\clients
```

8. Compile the Java Swing application using `ant` with the existing `build.xml` file:

```
ant -f build.xml compile.client
```

9. Run the application:

```
ant -f build.xml run
```

10. In the application, enter a SSN number of `123456789` and click Submit; if the MedRec application is deployed and running correctly, you will see information returned about a patient. The command window from which you ran the application shows the SOAP request and response messages resulting from the Web Service operation invokes.

When you run the stand-alone client application, make sure its CLASSPATH includes the client JAR file generated by the `clientgen` Ant task, as well as the runtime Web Services JAR file *`WL_HOME`*`/server/lib/webserviceclient.jar`, where *`WL_HOME`* refers to the top-level directory of WebLogic Platform.

Procedure 3: Invoke a Web Service from a .NET client.

You can also invoke the `MedRecWebServices` WebLogic Web Service from a .NET client application written in C#.

You must install the .NET Framework on your computer before you can create and run the .NET client. For details, see http://msdn.microsoft.com/netframework/downloads/howtoget.asp.

The sample .NET client that invokes the `MedRecWebServices` WebLogic Web service is in the following directory:

```
c:\medrec_tutorial\src\clients\CSharpClient
```

To run the client, execute the following file:

```
c:\medrec_tutorial\src\clients\CSharpClient\bin\Release\CSharpClient.exe
```

Best Practices

- When writing a Java client application to invoke a Web Service (either WebLogic or non-WebLogic), use the `clientgen` Ant task to generate the client JAR file that contains the JAX-RPC stubs for your Web Service. This client JAR file contains almost all the Java code you need to invoke a Web Service. Be sure to update the CLASSPATH of the client application with this JAR file.
- Stand-alone Java client applications also need to include the runtime client JAR file *`WL_HOME`*`\server\lib\webserviceclient.jar` in their CLASSPATH, where *`WL_HOME`* refers to the top-level directory of WebLogic Platform. This runtime JAR file contains the runtime implementation of JAX-RPC.
- Use the `wsdl` attribute of `clientgen` to generate the client JAR file from the WSDL, or public contract, of a Web Service.

The Big Picture

Client applications that invoke Web Services can be written using any technology: Java, Microsoft SOAP Toolkit, Microsoft .NET, and so on. Java client applications use the Java API for XML-Based RPC (JAX-RPC), a Sun Microsystems specification that defines the Java client API for invoking a Web Service. A Java client application can be an EJB deployed on WebLogic Server, or a stand-alone Java client.

In Tutorial 11: Exposing a Stateless Session EJB as a Web Service, you learned how to create and deploy the `MedRecWebServices` Web Service (part of the main MedRec application), which contains operations to find and update patient information, such as `updatePatient` and `findPatientBySsn`. The public contract of the Web Service is published via its WSDL, which lists its operations, the URL endpoints, and so on.

The Physician application, in a real-life situation, would be deployed on a separate WebLogic Server instance from the main MedRec application. The `PhysicianSessionEJB`, therefore, needs a way to communicate with the MedRec application over the Internet; using the operations of the `MedRecWebServices` Web Service is the ideal way to do this. The client JAR file generated by the `clientgen` Ant task contains the JAX-RPC stubs needed to invoke the Web Service operations—the amount of code you need to actually write in the EJB is very small.

The stand-alone Java client works almost the same as the EJB, except that the stand-alone client also needs the Web Services runtime client JAR file in its CLASSPATH; the EJB uses the runtime files contained in WebLogic Server.

Related Reading

- *Programming WebLogic Web Services*
- *Programming WebLogic XML*
- *Programming WebLogic Enterprise Java Beans*
- Java API for XML-Based RPC (JAX-RPC) 1.0 Specification
- Simple Object Access Protocol (SOAP) 1.1 Specification
- Web Services Description Language (WSDL) 1.1 Specification
- Java Swing

1 Building the MedRec Applications

Tutorial 13: Compiling the Entire MedRec Project

Previous tutorials explained how to compile, build, and deploy parts of individual MedRec applications. In this tutorial, you compile and build the entire MedRec application suite using the project-level `build.xml` file. Compiling the entire application suite is necessary to deploy all components on your system and verify that MedRec is running and usable.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial, create the project directory using the instructions in Tutorial 5: Creating the MedRec Project Directory.

Procedure

The project directory contains a master `build.xml` script that compiles all of the MedRec applications in the correct order. To run this script:

1. Open a command shell and set the development environment:

   ```
   c:\bea\user_projects\domains\medrecdomain\setenv.cmd
   ```

2. Move to the `src` subdirectory of the MedRec project directory:

   ```
   cd c:\medrec_tutorial\src
   ```

3. Use the Ant command to execute the master `build.xml` script with the `deploy.dev` target:

   ```
   ant deploy.dev
   ```

   The build process displays messages indicating the progress for each application. The entire build process take approximately 2 to 5 minutes to complete, depending on the speed of your computer. The script should complete with the message, "BUILD SUCCESSFUL."

Best Practices

- Not all projects require a master build script. If you are creating only a single Enterprise Application or a single component of an Enterprise Application, a single `build.xml` file using the WebLogic ant tasks will suffice.
- If your project requires multiple Enterprise Applications to be compiled in a particular sequence (because of shared utility classes or Web Services dependencies), use a master `build.xml` file at the source level to iterate through each application's `build.xml` files.

The Big Picture

The MedRec application suite has many dependencies that require coordination during the build process. When you run the master build file, the following events occur:

1. The contents of `startupEar` are compiled using the `wlcompile` task.
2. The contents of `common` are compiled. The `common` directory contains the source for several kinds of objects used by different MedRec applications:
   - Utility classes—constants used throughout the application suite, exceptions, factories, and the `ServiceLocator` class. Servlets in the Web Tier of the MedRec application suite use `ServiceLocator` to lookup generic services such as Enterprise JavaBeans.
   - Value objects—classes that represent data passed between tiers of the MedRec suite.
   - Action classes—classes used by the struts framework to control page flow in the Web tier of the MedRec suite.
   - JavaBeans—component beans used in the Web tier.
3. The `medrecEar` Enterprise Application is compiled. Although `medrecEar` uses the split development directory structure and the WebLogic Ant tasks in the build script, the application has several internal dependencies that are hard-coded in its `build.xml` script, using the `include` and `exclude` options to `wlcompile`.
4. The `physicianEar` application is compiled. The `physicianEar` Web Service client is generated from the `.wsdl` file copied into the `dist` directory.
5. The MedRec application suite client applications are compiled.

Related Reading

- Developing WebLogic Server Applications

Moving to Production Mode

Tutorial 14: Packaging MedRec for Distribution

In previous tutorials you configured, compiled, and deployed MedRec in a split-directory development environment. This tutorial describes how to use an Ant script to package the compiled Physician application into a single portable EAR that you can hand off to a production team.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Complete tutorials 5 through 9 to create the project directory and perform the intermediate build steps for the Physician Application. If you skipped any of tutorials 6 through 9, you can catch up by setting your environment:

```
c:\bea\user_projects\domains\medrecdomain\setenv.cmd
```

and then moving to the `c:\medrec_tutorial\src\physicianEar` subdirectory and executing the ant command:

```
ant -f build.xml
```

- Complete Tutorial 13: Compiling the Entire MedRec Project.

Procedure

- Step 1: Package the Physician application as an EAR.
- Step 2: Test the package.

Step 1: Package the Physician application as an EAR.

The following procedures create and run a script that packages the contents of the Physician application from the directories used in the split-directory development environment—`src` and `build`—into a single deployable, distributable EAR file in a distribution directory, `dist`.

1. Open a command shell and set your environment:

   ```
   c:\bea\user_projects\domains\medrecdomain\setenv.cmd
   ```

2. Move to the `src\physicianEar` subdirectory of the MedRec project directory:

   ```
   cd c:\medrec_tutorial\src\physicianEar
   ```

3. Use a text editor to create a new file called `package.xml`:

   ```
   notepad package.xml
   ```

   Note: If you do not want to create the `package.xml` file manually in this tutorial, copy the file named `wlpackage_tutorial.xml` to the new name, `package.xml`, and skip to step 9.

4. In the `package.xml` file, define a project named `tutorial` and supply a default target name:

   ```
   <project name="tutorial" default="package">
   ```

5. Define an `Ant` target name that you will specify when you run the script:

   ```
   <target name="package">
   ```

6. Provide the argument of the Ant target, which calls the `wlpackage` Ant task and combines the contents of the `src` and `build physicianEAR` directories into a single directory in `dist`.

```
   <wlpackage srcdir="c:/medrec_tutorial/src/physicianEar"
   destdir="c:/medrec_tutorial/build/physicianEar"

   toFile="c:/medrec_tutorial/dist/wlpackage_tutorial.ear" />

</target>
```

   See Split Development Directory Ant Tasks for more information about the `wlpackage` task.

7. Complete the `package.xml` file by closing the project element:

```
</project>
```

8. Your file contents should now resemble the following:

```
<project name="tutorial" default="package">

   <target name="package">

   <wlpackage srcdir="c:/medrec_tutorial/src/physicianEar"
   destdir="c:/medrec_tutorial/build/physicianEar"

   toFile="c:/medrec_tutorial/dist/wlpackage_tutorial.ear" />

   </target>

</project>
```

   Save the file and exit your text editor.

9. In the same command shell, enter the command to execute the build script:

```
ant -f package.xml
```

   You should receive the following output from the `package` task:

```
Buildfile: package.xml

package:

[jar] Building jar:
C:\medrec_tutorial\dist\wlpackage_tutorial.ear

BUILD SUCCESSFUL

Total time: 4 seconds
```

   If you do not receive the above output, `MedRecServer` may not have finished starting up, or you may have made a typo in creating the `package.xml` file. If

this occurs, wait until the server has finished starting up, and try to package using the installed tutorial file:

```
ant -f wlpackage_tutorial.xml
```

10. To verify that `wlpackage_tutorial.ear` has been created, change to `C:\medrec_tutorial\dist`:

    ```
    cd C:\medrec_tutorial\dist
    ```

    and then run `ls` or `dir`.

    ```
    dir wlpackage_tutorial.ear
    ```

11. Verify the contents of `wlpackage_tutorial.ear` using the jar command:

    ```
    C:\medrec_tutorial\dist>jar tf wlpackage_tutorial.ear
    ```

    You should see the following list of files and subdirectories:

    ```
    META-INF/
    META-INF/MANIFEST.MF
    APP-INF/
    APP-INF/classes/
    APP-INF/lib/
    physicianWebApp/
    physicianWebApp/WEB-INF/
    physicianWebApp/WEB-INF/classes/
    physicianWebApp/WEB-INF/classes/com/
    physicianWebApp/WEB-INF/classes/com/bea/
    physicianWebApp/WEB-INF/classes/com/bea/medrec/
    physicianWebApp/WEB-INF/classes/com/bea/medrec/utils/
    physicianWebApp/WEB-INF/src/
    physicianWebApp/WEB-INF/src/com/
    physicianWebApp/WEB-INF/src/com/bea/
    physicianWebApp/WEB-INF/src/com/bea/medrec/
    physicianWebApp/WEB-INF/src/com/bea/medrec/actions/
    physicianWebApp/WEB-INF/src/com/bea/medrec/utils/
    physSessionEjbs/
    ```

```
physSessionEjbs/com/
physSessionEjbs/com/bea/
physSessionEjbs/com/bea/medrec/
physSessionEjbs/com/bea/medrec/controller/
META-INF/application.xml
physicianWebApp/Confirmation.jsp
physicianWebApp/CreateRx.jsp
physicianWebApp/CreateVisit.jsp
physicianWebApp/Error.jsp
physicianWebApp/Login.jsp
physicianWebApp/PatientHeader.jsp
physicianWebApp/PhysicianHeader.jsp
physicianWebApp/Search.jsp
physicianWebApp/SearchResults.jsp
physicianWebApp/stylesheet.css
physicianWebApp/ViewProfile.jsp
physicianWebApp/ViewRecord.jsp
physicianWebApp/ViewRecords.jsp
physicianWebApp/WEB-INF/app.tld
physicianWebApp/WEB-INF/classes/com/bea/medrec/utils/Applicatio
nResources.properties
physicianWebApp/WEB-INF/classes/com/bea/medrec/utils/Applicatio
nResources_ja.properties
physicianWebApp/WEB-INF/struts-bean.tld
physicianWebApp/WEB-INF/struts-config.xml
physicianWebApp/WEB-INF/struts-html.tld
physicianWebApp/WEB-INF/struts-logic.tld
physicianWebApp/WEB-INF/struts-nested.tld
physicianWebApp/WEB-INF/struts-template.tld
physicianWebApp/WEB-INF/web.xml
physicianWebApp/WEB-INF/weblogic.xml
```

```
physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB.e
jb

webservices_tutorial.xml

wlcompile_tutorial.xml

package.xml

physicianWebApp/WEB-INF/lib/

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/

physicianWebApp/WEB-INF/classes/jsp_servlet/

physSessionEjbs/META-INF/

APP-INF/lib/exceptions.jar

APP-INF/lib/log4j-1.2.4.jar

APP-INF/lib/utils.jar

APP-INF/lib/value.jar

APP-INF/lib/webServicesEjb_client.jar

physicianWebApp/WEB-INF/lib/commons-beanutils.jar

physicianWebApp/WEB-INF/lib/commons-collections.jar

physicianWebApp/WEB-INF/lib/commons-dbcp.jar

physicianWebApp/WEB-INF/lib/commons-digester.jar

physicianWebApp/WEB-INF/lib/commons-logging.jar

physicianWebApp/WEB-INF/lib/commons-pool.jar

physicianWebApp/WEB-INF/lib/commons-services.jar

physicianWebApp/WEB-INF/lib/commons-validator.jar

physicianWebApp/WEB-INF/lib/commonWeb.jar

physicianWebApp/WEB-INF/lib/log4j-1.2.4.jar

physicianWebApp/WEB-INF/lib/struts.jar

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/CreateRx
Action.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysBase
LookupDispatchAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/CreateVi
sitAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysBase
Action.class
```

```
physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysLog4
jInit.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysLogi
nAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysLogo
utAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysView
ProfileAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysView
RecordAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/PhysView
RecordsSummaryAction.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/SearchAc
tion.class

physicianWebApp/WEB-INF/classes/com/bea/medrec/actions/SearchRe
sultsAction.class


physicianWebApp/WEB-INF/classes/com/bea/medrec/utils/PhysConsta
nts.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__physicianheader.c
lass

physicianWebApp/WEB-INF/classes/jsp_servlet/__confirmation.clas
s

physicianWebApp/WEB-INF/classes/jsp_servlet/__createrx.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__createvisit.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__error.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__login.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__patientheader.cla
ss

physicianWebApp/WEB-INF/classes/jsp_servlet/__search.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__searchresults.cla
ss

physicianWebApp/WEB-INF/classes/jsp_servlet/__viewprofile.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__viewrecord.class

physicianWebApp/WEB-INF/classes/jsp_servlet/__viewrecords.class
```

```
physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionHome.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSession.class

physSessionEjbs/com/bea/medrec/controller/PhysicianClientUtils.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_Intf.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_Impl.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_EOImpl.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_HomeImpl.class

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_HomeImplRTD.xml

physSessionEjbs/com/bea/medrec/controller/PhysicianSessionEJB_n7enxc_EOImplRTD.xml

physSessionEjbs/ejbgen-build.xml

physSessionEjbs/META-INF/weblogic-ejb-jar.xml

physSessionEjbs/META-INF/ejb-jar.xml

physSessionEjbs/_WL_GENERATED
```

The `EAR` file you have created contains the Physician application bundled into a deployable archive.

Step 2: Test the package.

To confirm that the archive is deployable, use the Administration Console Deployment Assistant to deploy it to `MedRecServer`.

1. With `MedRecServer` running, access the Administration Console at `http://localhost:7101/console`.
2. In the left-hand pane, expand Deployments and select Applications.
3. In the right-hand pane, select Deploy a new Application.

4. Use the Location links to select `C:\medrec_tutorial\dist`.
5. Select `wlpackage_tutorial.ear` and click Continue.
6. In the Deploy an Application page, verify that `MedRecServer` is the targeted server and click Deploy.
7. The Deploy panel shows the status of the deployment. It refreshes to update the status, and on completion shows the success or failure of the deployment.

Best Practices

For actual deployment for production, package your application in exploded, unarchived format. Doing so allows you to access and update files, for example deployment descriptor files, without having to unarchive and then rearchive the entire application. See Tutorial 14: Deploying the MedRec Package for Production for instructions on deploying MedRec in exploded format.

The Big Picture

In this tutorial, you packaged the Physician application into a single portable EAR file suitable for handing off to a production team. The split directory structure for development presents no obstacle to switching to a manageable single directory structure for production.

Related Reading

- Enterprise Application Deployment Descriptor Elements in *Developing WebLogic Server Applications*
- Overview of WebLogic Server Deployment in *Deploying WebLogic Server Applications*

1 Moving to Production Mode

Tutorial 15: Deploying the MedRec Package for Production

This tutorial describes how to use the Administration Console to deploy the MedRec application to a server for production. In this example the application files are packaged in exploded format in directories, rather than as EAR files. The advantage of the exploded format for production is that deployment descriptor files in an exploded directory can be updated without having to be unarchived and then rearchived following the update.

For instructions on packaging the MedRec application into a single archived EAR file, in contrast to the exploded format used in this tutorial, see Tutorial 14: Packaging MedRec for Distribution. The advantage of packaging into an EAR file is that the application is more portable when bundled into a single file, and can more easily be moved or distributed.

The procedures below deploy the exploded contents of the `medrecEar`, `startupEar`, and `physicianEar` subdirectories of the `dist` directory, created in Tutorial 13: Compiling the Entire MedRec Project.

- `medrecEar` is MedRec's main enterprise application, containing its patient and administrative Web Applications, the Web service used by the physician Web application, and the EJBs that store and run MedRec's logic and data.

- `physicianEar` is a separate component of the MedRec application, with a different set of users, which communicates with `medrecEar` using a Web Service.
- `startupEar` contains a single class file that starts the browser when the servlets in the Web Applications are initialized.

For more information about the components of the MedRec application, see Overview of the Avitek Medical Records Development Tutorials.

The tutorial includes the following sections:

- Prerequisites
- Procedure
- Best Practices
- Related Reading

Prerequisites

Before starting this tutorial:

- Create the MedRec domain and the MedRec server. See Tutorial 1: Creating a WebLogic Domain and Server Instance for Development.
- Work through Tutorial 2: Starting the PointBase Development Database.
- Work through Tutorial 3: Setting Up WebLogic Server Resources for the MedRec Server.
- Create the MedRec project directory. See Tutorial 5: Creating the MedRec Project Directory.
- Familiarize yourself with how MedRec's split directory structure works, in Tutorial 7: Compiling Applications Using the Split Development Directory.
- Most importantly, work through Tutorial 13: Compiling the Entire MedRec Project, because the directories that contain the MedRec application in exploded format are created in its steps.

Procedure

1. Start the MedRec server, if it is not already running.

   From the Windows start menu:

   ```
   Start->Programs->BEA WebLogic Platform 8.1->User
   Projects->MedRecDomain->Start Server
   ```

   From the command line:

   ```
   C:\bea\user_projects\domains\MedRecDomain>startWebLogic.cmd
   ```

2. Open the Administration Console.

   Once the server starts, open `http://localhost:7101/console` in a browser, where `localhost` is the network name of your computer.

3. Undeploy applications from previous tutorials:

   a. Expand Deployments in the left pane of the Console and click the Applications folder. The right pane of the Console may show existing deployments from the previous tutorials (`medrec_deployment` and `wlpackage_tutorial`).

   b. Click the trash can icon to the right of an existing deployment. The Console prompts you to undeploy the application.

   c. Click Yes to remove the deployment, then click Continue.

   d. Repeat the above steps for any other MedRec deployments on the server.

4. Deploy the MedRec applications to `MedRecServer`:

   a. Expand Deployments in the left pane of the Console.

   b. Right-click Applications and select Deploy a New Application.

      This initiates the Enterprise Application Deployment Assistant in the right panel.

   c. Use the links in the Location field to navigate to `C:\medrec_tutorial\dist`.

      The Deploy an Application page table contains three applications that were created in Tutorial 12: `medrecEar`, `physicianEar`, and `startupEar`. Deploy all three applications, starting with `medrecEar`.

   d. Select `medrecEar` and click Continue.

e. Click Deploy.

   The Console displays the Deploy panel, which shows the deployment status of applications and deployment activities on the server. The table in this page shows that the deployment is underway, and then refreshes to report the success or failure of the deployment.

f. Use steps a to c to return to the Deploy an Application page, and select and deploy `physicianEar`.

g. Use steps a to c to return to the Deploy an Application page, and select and deploy `startupEar`.

5. Access the MedRec applications to confirm that they are deployed. In a browser, navigate to `http://127.0.0.1:7101/physician`, and log in using the username and password supplied in the text fields.

Best Practices

- Use the Administration Console Deployment Assistant to deploy your application in a graphical environment that shows you the choices you can make in your deployment, as an alternative to deploying using the command-line tool `weblogic.Deployer` or to editing Ant scripts that run deployment targets.
- For production, deploy in exploded format to simplify the process of updating the application.
- Use the Administration Console to monitor the progress of MedRec deployment and application activities. In case of errors, scroll up in the console text for useful messages.

Big Picture

The split-directory structure introduced in WebLogic Server 8.1 lets you deploy MedRec's compiled and generated files separately from the editable files. This capability is convenient during the development stage, when changes to the application are frequent. The expected format for production is the traditional single-directory structure, with the separate applications in exploded format in separate subdirectories.

In this tutorial, you deployed MedRec's applications from a directories containing the applications and all of their components and support files. The applications' exploded format makes their editable files more accessible than they would be if they were bundled into archives.

Each application subdirectory in `dist` contains both the compiled classes and generated deployment descriptors from the `build` directory, and the editable deployment descriptors and other files from the `src` directory.

Related Reading

- Enterprise Application Deployment Descriptor Elements in *Developing WebLogic Server Applications*
- Overview of WebLogic Server Deployment in *Deploying WebLogic Server Applications*

1 Moving to Production Mode

Tutorial 16: Using a Production Database Management System

This tutorial describes how to change the database used by the deployed MedRec application from one on a development relational database management system (PointBase) to a production DBMS (Oracle).

In particular, this tutorial shows you how to use the Administration Console to:

- Create both XA and non-XA JDBC connection pools used to connect to an Oracle database.
- Update the existing JDBC datasource used by the MedRec application to use the new Oracle XA JDBC connection pool.
- Update the existing JMS JDBC store to use the new Oracle non-XA JDBC connection pool.

Note: It is assumed that you have already installed and configured the Oracle database management system and that you have created an Oracle database. Describing how to perform these tasks is beyond the scope of this tutorial.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Install and configure the Oracle database management system. See the Oracle documentation.
- Create an Oracle database. See the Oracle documentation.
- Work through the MedRec tutorials up to Tutorial 15: Deploying the MedRec Package for Production, which describe how to compile, package, and deploy to the MedRec server the three applications that make up the MedRec application suite: `medrecEar`, `physicianEar`, and `startupEar`.

Procedure

Follow these steps:

- Step 1: Create the Oracle tables and populate with MedRec application data.
- Step 2: Invoke the Administration Console.
- Step 3: Create an Oracle XA JDBC connection pool.
- Step 4: Create a non-XA Oracle JDBC connection pool.
- Step 5: Update the MedRecTXDataSource to use the new Oracle XA connection pool.
- Step 6: Update the JMS JDBC store to use the new Oracle non-XA connection pool.
- Step 7: Shut down and restart the MedRec server.

- Step 8: Test the MedRec application using the Oracle database.

Step 1: Create the Oracle tables and populate with MedRec application data.

BEA provides two SQL scripts that you can use to create and populate the tables of your Oracle database:

- `medrec_oracle.ddl`—contains the SQL statements for creating the tables used by the MedRec application.
- `medrec_oracle.sql`—contains the SQL statements for populating the tables with data.

These scripts are located in *`SAMPLES_HOME`*`\server\medrec\setup\db` directory, where *`SAMPLES_HOME`* refers to the main examples directory of your WebLogic Server installation, such as `c:\beahome\weblogic81\samples`.

Note: It is beyond the scope of this tutorial to describe exactly how to create and populate Oracle tables. See the Oracle documentation.

Step 2: Invoke the Administration Console.

You use the Administration Console to create and update the WebLogic Server resources used by the MedRec application suite.

1. Enter the following URL in your browser:

   ```
   http://127.0.0.1:7101/console
   ```

2. Enter `weblogic` as the username and password, then click Sign In.

Step 3: Create an Oracle XA JDBC connection pool.

The JDBC connection pool configuration describes how to connect physically from WebLogic Server to a database, in this case an Oracle database. This procedure describes how to create a JDBC connection pool that uses an XA JDBC driver, which is a BEA best practice.

The procedure also shows how to specify support for SQL without global transactions.

1. In the left pane of the Administration Console, expand Services—JDBC.
2. Click Connection Pools.

3. In the right pane, click Configure a new JDBC Connection Pool.
4. For the Database Type, select `Oracle`.
5. For the Database Driver, select `Oracle's Driver (Thin XA) Versions: 8.1.7,9.0.1,9.2.0`.
6. Click Continue.
7. In the Name field, enter `MedRecPool-Oracle-XA`.
8. In the Database Name field, enter the name of your Oracle database.
9. In the Host Name field, enter the name of the computer that is hosting the Oracle database management system.
10. In the Port field, enter the port of the Oracle server.
11. In the Database User Name field, enter the name of the Oracle database user.
12. In the Password and Confirm Password fields, enter the password of the database user.
13. Click Continue.
14. Ensure that the information to test the connection to the Oracle database is correct, then click Test Driver Configuration.

    Note: Be sure you have started the Oracle database management system and that the database is accessible, or the test of its driver configuration will fail.
15. After verifying that the connection succeeded, click Create and Deploy.
16. In the left pane of the Administration Console, click MedRecPool-Oracle-XA under the Services→JDBC→Connection Pools node.
17. In the right pane, select the Configuration→Connections tab.
18. Click the Show link to the right of the Advanced Options label.
19. Scroll down to the end of the page and click Supports Local Transactions.
20. Click Apply.

Step 4: Create a non-XA Oracle JDBC connection pool.

This procedure describes how to create a JDBC connection pool that does not use an XA JDBC driver.

Typically you use an XA JDBC driver when creating a connection pool. However, because JMS JDBC stores do not support XA resource drivers (WebLogic JMS implements its own XA resource), you need an additional connection pool that is non-XA. Later procedures show how to associate the XA connection pool to a JDBC DataSource and the non-XA connection pool to a JMS JDBC store.

1. In the left pane of the Administration Console, expand Services—JDBC.
2. Click Connection Pools.
3. Click Configure a new JDBC Connection Pool.
4. For the Database Type, select `Oracle`.
5. For the Database Driver, select `Oracle's Driver (Thin) Versions: 8.1.7,9.0.1,9.2.0`.
6. Click Continue.
7. In the Name field, enter `MedRecPool-Oracle`.
8. In the Database Name field, enter the name of your Oracle database.
9. In the Host Name field, enter the name of the computer that is hosting the Oracle database management system.
10. In the Port field, enter the port of the Oracle Server.
11. In the Database User Name field, enter the name of the Oracle database user.
12. In the Password and Confirm Password fields, enter the password of the database user.
13. Click Continue.
14. Ensure that the information to test the connection to the Oracle database is correct, then click Test Driver Configuration.
15. After verifying that the connection succeeded, click Create and Deploy.

Step 5: Update the MedRecTXDataSource to use the new Oracle XA connection pool.

1. In the left pane of the Administration Console, expand Services—JDBC—Data Sources.
2. Click MedRecTxDataSource.
3. In the right pane, select `MedRecPool-Oracle-XA` in the Pool Name drop-down choice box.
4. Click Apply.

Step 6: Update the JMS JDBC store to use the new Oracle non-XA connection pool.

1. In the left pane of the Administration Console, expand Services—JMS—Stores.
2. Click MedRecJMSJDBCStore.
3. In the right pane, select `MedRecPool-Oracle` in the Connection Pool drop-down choice box.
4. Click Apply.

Step 7: Shut down and restart the MedRec server.

Changing the connection pool associated with a datasource requires that WebLogic Server be restarted so that the changes take effect. In a real-life situation, you would very likely restart WebLogic Server when moving from development to production mode anyway.

1. In the left pane of the Administration Console, expand the Servers node.
2. Right-click MedRecServer and choose Start/Stop This Server.
3. In the right pane, click Graceful Shutdown of this Server.
4. Click Yes.
5. Once the MedRec server shuts down, open a new command window and change to the MedRecDomain directory:

```
cd C:\bea\user_projects\domains\MedRecDomain
```

6. Restart the MedRec Server by executing its start script:

   ```
   startWebLogic.cmd
   ```

7. Invoke the Administration Console in a browser.

8. Verify that the three applications (`medrecEar`, `physicianEar`, and `startupEar`) are deployed by expanding, in the left pane, Deployments→Applications→*AppName*, then selecting the Deploy tab in the right pane. If the Module Status for any application is anything other than Active, click Deploy Application.

Step 8: Test the MedRec application using the Oracle database.

1. Shut down the PointBase database by closing the command window from which you started it. This step ensures that the application is unable to get data from the PointBase database.

2. In a browser, navigate to `http://127.0.0.1:7101/physician`, and log in using the username and password supplied in the text fields.

3. Enter `Couples` in the Last Name field and click Search. If you see an entry for Fred Couples, the data has come from your Oracle database.

Best Practices

- Use JDBC DataSources to separate the details about connecting to a database from an application. This makes it easy to change the database to which an application connects without having to update the application itself.
- WebLogic Server uses JDBC drivers to create the physical database connection in a connection pool. When using the Administration Console to create a JDBC connection pool, you specify the driver you want to use. BEA offers the following drivers to connect to an Oracle database:
  - WebLogic jDriver for Oracle (Type 2, which requires native libraries)
  - WebLogic jDriver for Oracle XA (Type 2, which requires native libraries)
  - Oracle Thin driver (Type 4, pure Java)

- Oracle Thin XA driver (Type 4, pure Java)

The WebLogic jDrivers use native libraries, but may perform faster than the pure Java Type 4 drivers from Oracle. For additional details about deciding which driver to use in your connection pool, see Introduction to WebLogic JDBC.

The Big Picture

A JDBC DataSource makes it easy to change the database management system to which a WebLogic Server application connects because the DataSource provides a layer of abstraction between the application and the details of a connection to the database.

One DataSource is associated with one JDBC connection pool, which describes the details about how to connect to a database, such as the host and port of the database server, the name of the database, the database user, and so on. The deployment descriptor of the component that needs database access, such as an entity EJB, lists the DataSource that it will use to connect to a database. Therefore, to change the database to which an application connects, you simply create a new connection pool, and use the Administration Console to update the DataSource, changing the connection pool to which it is associated.

For example, `AddressEJB` is an entity EJB in the medrecEar application. It uses container-managed persistence (CMP) to persist its data to a database. The WebLogic-specific deployment descriptor file that contains CMP information about `AddressEJB`, `weblogic-cmp-rdbms-jar.xml`, specifies that the EJB uses the `MedRecTxDataSource` when connecting to a database, as shown in the following excerpt:

```
<weblogic-rdbms-jar>
  <weblogic-rdbms-bean>
    <ejb-name>AddressEJB</ejb-name>
    <data-source-name>MedRecTxDataSource</data-source-name>
    ...
  </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

The deployment descriptor does not include specific details about how to connect to the database, making the application more portable.

Changing the database to which the JMS JDBC store persists data is very similar: you use the Administration Console to change the connection pool to which the store is associated.

You must restart WebLogic Server after making these changes to ensure that all connections to the old database are ended and the application starts connecting to the new database.

Related Reading

- Oracle documentation
- *Introduction to WebLogic JDBC* at http://e-docs.bea.com/wls/docs81/jdbc/intro.html
- *WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs81/oracle/index.html
- *JDBC Connection Pools* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html
- *WebLogic JMS Fundamentals* at http://e-docs.bea.com/wls/docs81/jms/fund.html
- *Configuring JMS* at http://e-docs.bea.com/wls/docs81/ConsoleHelp/jms_config.html

1 Moving to Production Mode

Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console

This tutorial describes how to secure application and URL (Web) resources using the Administration Console. It includes step-by-step procedures for creating users, groups, and global security roles. It also provides procedures for creating security policies at various levels in the application and URL (Web) resource hierarchies.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create `MedRecDomain` and `MedRecServer`, and start `MedRecServer`. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development" on page -1.
- Deploy the enterprise application named `MedRecEar`. See "Tutorial 15: Deploying the MedRec Package for Production" on page 1-1.
- Read the following sections in *Securing WebLogic Resources*:
  - Types of WebLogic Resources
  - Techniques for Securing URL (Web) and EJB Resources
  - Prerequisites for Securing URL (Web) and EJB Resources
  - Types of Security Roles: Global Roles and Scoped Roles

Procedure

Follow these steps to secure application and URL (Web) resources using the Administration Console:

Step 1: Specify security realm settings.

1. Start a Web browser and type `http://localhost:7101/console/`.
2. Enter `weblogic` as the username and `weblogic` as the password, then click Sign In to sign in to the Administration Console for `MedRecServer`.
3. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
4. Click the `myrealm` security realm.
5. On the General tab, from the Check Roles and Policies drop-down menu, select All Web Applications and EJBs.

   This setting means that the WebLogic Security Service will perform security checks on *all* URL (Web) and EJB resources. For more information, see Understanding How to Check Security Roles and Security Policies in *Securing WebLogic Resources*.
6. From the On Future Redeploys drop-down menu, select Ignore Roles and Policies From DD.

   This setting means that you will set security for Web Application and EJB resources in the Administration Console, not in deployment descriptors. For more information, see Understanding What to Do on Future Redeploys of the WebLogic Resource in *Securing WebLogic Resources*.
7. Click Apply to save your changes.
8. Restart `MedRecServer`. (See Starting and Stopping Servers: Quick Reference in *Configuring and Managing WebLogic Server*.)

Step 2: Create groups.

1. In the navigation tree at the left side of the Administration Console, expand Security->Realms->myrealm.
2. Click Groups.

   The Groups page displays all groups currently defined in the WebLogic Authentication provider's database.

3. Click the Configure a new Group link to display the Create Group page.
4. On the General tab, in the Name field, type `MedRecAdmins`.
5. In the Description field, type `MedRecAdmins can log on to the MedRec Administrators website`.
6. Click Apply to save your changes.
7. Repeat steps 4 - 7 to create a group named `MedRecPatients`, with a description of `MedRecPatients can log on to the MedRec Patients website`.
8. In the navigation tree, click Groups, and confirm that the groups have been added.

   The Groups page shows the groups added to the WebLogic Authentication provider's database.

Step 3: Create users and add the users to groups.

1. In the navigation tree, click Users.

   The Users page displays all users currently defined in the WebLogic Authentication provider's database.
2. Click the Configure a new User link to display the Create User page.
3. On the General tab, in the Name field, type `admin@avitek.com`.
4. In the Description field, type `MedRec administrator`.
5. In the Password and Confirm Password fields, type `weblogic`.
6. Click Apply to save your changes.
7. Select the Groups tab.
8. In the Possible Groups list box, highlight the `MedRecAdmins` group.
9. Click the highlighted arrow to move the `MedRecAdmins` group from the Possible Groups list box to the Current Groups list box.
10. Click Apply to save your changes.

11. Repeat steps 2 - 10 to create a user named `larry@celtics.com`, a `MedRec patient` who also uses the `weblogic` password and belongs in the `MedRecPatients` group.
12. In the navigation tree, click Users, and confirm that the users have been added.

    The Users page shows the users added to the WebLogic Authentication provider's database.

Step 4: Create global roles and grant the global roles to the groups.

1. In the navigation tree, click Global Roles.

   The Global Roles page displays all global roles currently defined in the WebLogic Role Mapping provider's database.
2. Click the Configure a new Global Role link to display the Create Global Role page.
3. On the General tab, in the Name field, type `MedRecAdmin`.
4. Click Apply to save your changes.
5. Select the Conditions tab.
6. In the Role Condition list box, highlight `Caller is a Member of the Group`.
7. Click Add to display the Groups window.
8. In the Enter Group Name field, type `MedRecAdmins`.
9. Click Add, then click OK.

   The Groups window closes. The Role Statement list box reads:

   ```
   Caller is a Member of the Group
      MedRecAdmins
   ```
10. Click Apply to save your changes.
11. Repeat steps 2 - 11 to create a global role named `MedRecPatient` and to grant this global role to the `MedRecPatients` group.
12. In the navigation tree, click Global Roles, and confirm that the global roles have been added.

The Global Roles page shows the global roles added to the WebLogic Role Mapping provider's database.

Step 5: Secure the MedRecEAR application.

1. In the navigation tree, expand Deployments->Applications.
2. Right-click `MedRecEAR`.
3. From the menu, select Define Security Policy to display the Policy Editor page.

   Selecting this option enables you to create a security policy that will encompass *all* components in the deployed Enterprise Application.
4. In the Policy Condition list box, highlight `Caller is Granted the Role`.
5. Click Add to display the Roles window.
6. In the Enter Role Name field, type `MedRecAdmin`.
7. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      MedRecAdmin
   ```
8. Click Apply to save your changes.

Step 6: Attempt to access a JSP in the MedRecEAR application.

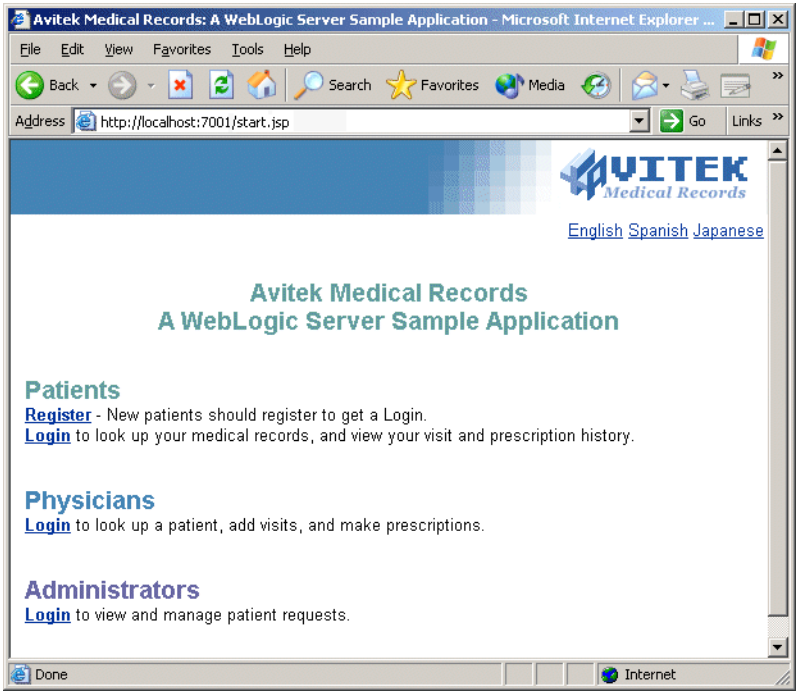
1. Open a new Web browser and type `http://localhost:7101/start.jsp`.

   The browser prompts you for a username and password.
2. In the username field, type `larry@celtics.com`, and in the password field, type `weblogic`, then click OK.

   The browser re-prompts you for a username and password.
3. In the username field, type `admin@avitek.com`, and in the password field, type `weblogic`, then click OK.

   The browser displays the page shown in Figure 1.

Figure 1: Avitek Medical Records Start Page



User `larry@celtics.com` was denied access because you previously secured all components of `MedRecEAR` (including `start.jsp`, part of `MainWAR`) with a security policy based on the global security role `MedRecAdmin`, which user `admin@avitek.com` is granted but user `larry@celtics.com` is not.

Step 7: Secure the Patient Web Application.

1. In the navigation tree at the left side of the Administration Console, expand Deployments->Applications->MedRecEAR, then right-click on the `patient` Web Application.
2. From the menu, select Define Security Policy to display the General tab.

   Selecting this option enables you to create a security policy for this particular Web Application or a particular component within the Web Application.
3. In the URL Pattern field, type: `/*`

   The URL pattern of `/*` will secure all components, including JSPs and servlets.

4. Click the Define Security Policy button to display the Policy Editor page.
5. In the Policy Condition list box, highlight `Caller is Granted the Role`.

   Do *not* modify the value shown in the Methods drop-down menu. (It should read: `ALL`.)
6. Click Add to display the Roles window.
7. In the Enter Role Name field, type `Anonymous`.

   Unlike the `MedRecAdmin` and `MedRecPatient` global roles you created and used in previous steps, the `Anonymous` role is a default global role that is predefined in WebLogic Server.
8. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      Anonymous
   ```

   By defining this security policy on `PatientWAR`, you are overriding the security policy that has already been defined for all components of the `MedRecEAR` in "Step 5: Secure the MedRecEAR application.." Specifically, you are overriding the inherited policy statement of:

   ```
   Caller is Granted the Role
      MedRecAdmin
   ```

   that is shown in the Inherited Policy Statements list box.

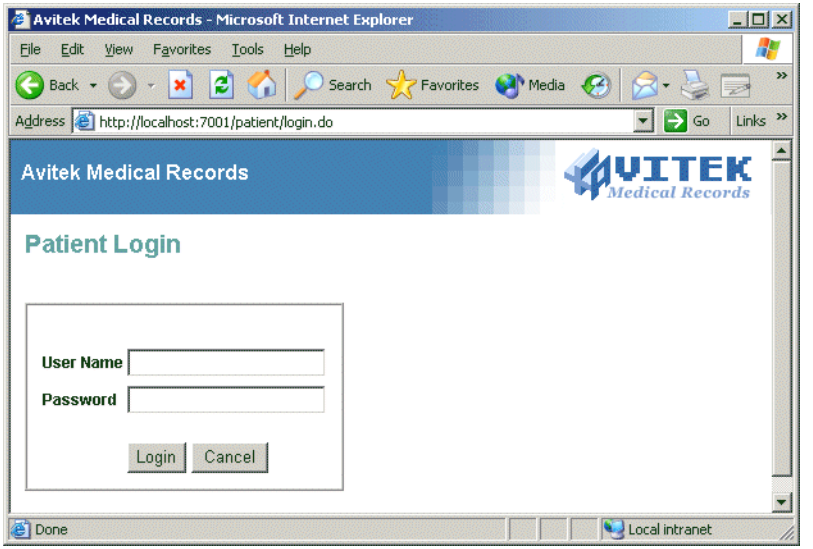
   By overriding the security policy to grant access to users in the `Anonymous` global role (rather than the `MedRecAdmin` global role), you are actually making access on these pages *less* restrictive. (All users are granted the `Anonymous` global role.)
9. Click Apply to save your changes.

Step 8: Attempt to access a JSP in the PatientWAR.

Open a new Web browser and type `http://localhost:7101/patient/login.do`.

The browser displays the page shown in Figure 2:.

Figure 2: Patient Login Page



This page is displayed because you secured all components of `PatientWAR` with a security policy based on the global security role `Anonymous`, a security role that all users are granted. Therefore, no login is required to access the `login.do` page. (The user name and password fields are shown because of the `login.do` page's design.)

Step 9: Secure the medicalrecord.do page.

1. In the navigation tree at the left side of the Administration Console, right-click on the `patient` Web Application.
2. From the menu, select Define Security Policy to display the General tab.

   Selecting this option enables you to create a security policy for this particular Web Application or a particular component within the Web application.

   Notice that the URL pattern you typed in "Step 7: Secure the Patient Web Application." on page 1-7 appears as a link under the title: Already Defined URL Patterns. This allows you to modify existing security policies more easily.
3. In the URL Pattern field, type `medicalrecord.do`.

   Because you are creating the security policy on `PatientWAR`, the context path of `/patient` is implied in the URL pattern. (WebLogic Server obtains this context path from the Web Application's deployment descriptor.)

4. Click the Define Security Policy button to display the Policy Editor page.
5. In the Policy Condition list box, highlight `Caller is Granted the Role`.

   Do *not* modify the value shown in the Methods drop-down menu. (It should read `ALL`.)
6. Click Add to display the Roles window.
7. In the Enter Role Name field, type `MedRecPatient`.
8. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      MedRecPatient
   ```

   By defining this security policy on `medicalrecord.do`, you are overriding the security policy that has already been defined for all components within `PatientWAR` in "Step 7: Secure the Patient Web Application.." Specifically, you are overriding the inherited policy statement of:

   ```
   Caller is Granted the Role
      Anonymous
   ```

   that is shown in the Inherited Policy Statements list box.
9. Click Apply to save your changes.

Step 10: Attempt to access the medicalrecord.do page.

1. Open a new Web browser and type `http://localhost:7101/patient/medicalrecord.do`.

   The browser redirects you to the login page shown in Figure 2:. This result occurs because only users who are granted the `MedRecPatient` global role can access the `medicalrecord.do` page, but all users (who are granted the `Anonymous` global role) can still access `login.do`.
2. In the username field, type `admin@avitek.com`, and in the password field, type `weblogic`, then click Login.

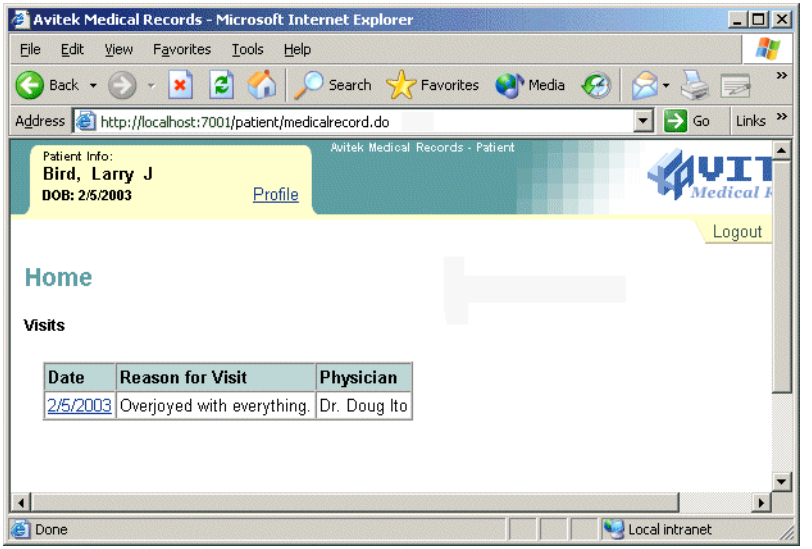
   The browser redisplays the login page shown in Figure 2: and indicates that you have entered an invalid username and/or password. This result occurs because only users who are granted the `MedRecPatient` global role can access the

medicalrecord.do page, and user admin@avitek.com is granted the global role MedRecAdmin.

3. In the username field, type larry@celtics.com, and in the password field, type weblogic, then click Login.

   The browser redirects you to Larry's Medical Records page, shown in Figure 3:. This result occurs because user larry@celtics.com is granted the MedRecPatient global role, which is required to access the medicalrecord.do page.

Figure 3: Larry's Medical Records Page



Best Practices

- The security realm settings are extremely important. If you want to secure URL (Web) resources using the WebLogic Server Administration Console rather than deployment descriptors, you must use the Check Roles and Policies/On Future Redeploys combination specified in "Step 1: Specify security realm settings.."
- If you have deployed an application (or module) with the On Future Redeploys drop-down menu set to Ignore Roles and Policies From DD one or more times

before setting it to Initialize Roles and Policies From DD, you can still set security policies and security roles using the Administration Console. These changes will override any security specified in deployment descriptors.

- Do not use blank spaces, commas, hyphens, or any characters in this comma-separated list for user, group, or security role names: \t, < >, #, |, &, ~, ?, ( ), { }. User, group, and security role names are case sensitive. The proper syntax for a security role name is as defined for an `Nmtoken` in the Extensible Markup Language (XML) recommendation. The BEA convention is that group names are plural, and security role names are singular.
- BEA recommends assigning users to groups, then creating role statements using the `Caller is a Member of the Group` role condition. Individual users could also be granted a security role, but this is a less typical practice.
- BEA recommends using security roles (rather than users or groups) to secure WebLogic resources. Following this process makes it more efficient for administrators who work with large numbers of users.
- Create policy statements based on your organization's established business procedures.
- When creating new security policies, look for policy statements in the Inherited Policy Statement box of the Policy Editor page. If inherited policy statements exist, you will be overriding them.
- Remember that more-specific security policies override less-specific security policies. For example, a security policy on a specific URL pattern in a Web application overrides a security policy on the Web application. Take care when overriding with less restrictive security policies (that is, giving a wider set of users access to a smaller set of components or WebLogic resources).
- Take care to ensure that you understand the security policies and the security role mappings on each URL pattern. If there are any URL patterns that you do not expect, be sure to investigate.
- Be sure you understand the precedence of servlet mappings to URL patterns as specified in Chapter 11 of the Servlet 2.3 specification. This describes which URL pattern will have precedence when an URL matches multiple URL patterns.
- You can delete all security settings for an application (or module) by deleting it entirely from the WebLogic Server domain and then redeploying it.

The Big Picture

This tutorial shows you how to secure application and various URL (Web) resources using some examples. These examples may or may not be different from those used in the full MedRec application. However, the full MedRec application uses these same principles (as well as programmatic security) to secure URL (Web) resources for both MedRec administrators and patients.

Related Reading

- *Securing WebLogic Resources*
- "Tutorial 18: Securing Enterprise JavaBean (EJB) Resources Using the Administration Console" on page 1-1
- "Tutorial 19: Copying and Reinitializing Security Configurations" on page 1-1

1 Moving to Production Mode

Tutorial 18: Securing Enterprise JavaBean (EJB) Resources Using the Administration Console

This tutorial describes how to secure Enterprise JavaBean (EJB) resources using the Administration Console. It includes step-by-step procedures for creating scoped roles and security policies at various levels in the EJB resource hierarchy.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create `MedRecDomain` and `MedRecServer`, and start `MedRecServer`. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development" on page -1.
- Deploy the Enterprise Application named `MedRecEar`. See "Tutorial 15: Deploying the MedRec Package for Production" on page 1-1.
- If you did not complete "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console," create the users and groups described in "Step 2: Create groups." and "Step 3: Create users and add the users to groups." sections of that tutorial.
- Read the following sections in *Securing WebLogic Resources*:
  - Types of WebLogic Resources
  - Techniques for Securing URL (Web) and EJB Resources
  - Prerequisites for Securing URL (Web) and EJB Resources
  - Types of Security Roles: Global Roles and Scoped Roles

Procedure

Follow these steps to secure Enterprise JavaBean (EJB) resources using the Administration Console:

Step 1: Specify security realm settings.

Note: If you completed this step as part of "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console," you can skip to "Step 2: Create scoped roles and grant the scoped roles to groups." on page 1-4.

1. Start a Web browser and type `http://localhost:7101/console/`.
2. Enter `weblogic` as the username and `weblogic` as the password, then click Sign In to sign in to the Administration Console for the `MedRecServer`.
3. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
4. Click the `myrealm` security realm.
5. On the General tab, from the Check Roles and Policies drop-down menu, select All Web Applications and EJBs.

   This setting causes the WebLogic Security Service to perform security checks on *all* URL (Web) and EJB resources. For more information, see Understanding How to Check Security Roles and Security Policies in *Securing WebLogic Resources*.
6. From the Future Redeploys drop-down menu, select Ignore Roles and Polices From DD.

   This setting indicates that you will set security for Web application and EJB resources using the Administration Console, not deployment descriptors. For more information, see Understanding What to Do on Future Redeploys of the WebLogic Resource in *Securing WebLogic Resources*.
7. Click Apply to save your changes.
8. Restart `MedRecServer`. (For help, see Starting and Stopping Servers: Quick Reference in the *Configuring and Managing WebLogic Server*.)

Step 2: Create scoped roles and grant the scoped roles to groups.

1. In the navigation tree, expand Deployments->Applications->MedRecEar.
2. Right-click `sessionEjbs`.
3. From the menu, select Define Scoped Role to display the Scoped Roles page.

   This page displays all the scoped roles currently defined in the WebLogic Role Mapping provider's database.

   Selecting this option enables you to create a security role that is scoped to this particular EJB JAR. Thereafter, the scoped role can be used in a security policy for this EJB JAR.
4. Click the Configure a new Scoped Role link to display the Create Scoped Role page.
5. On the General tab, in the Name field, type `MedRecSessionEJBPatient`.
6. Click Apply to save your changes.
7. Select the Conditions tab.
8. In the Role Condition list box, highlight `Caller is a Member of the Group`.
9. Click Add to open the Groups window.
10. In the Enter Group Name field, type `MedRecPatients`.

    Note: You created the `MedRecPatients` group as part of "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console." Recall that user `larry@celtics.com` is the only user in this group.
11. Click Add, then click OK.

    The Groups window closes. The Role Statement list box reads:

    ```
    Caller is a Member of the Group
       MedRecPatients
    ```
12. Click Apply to save your changes.
13. In the navigation tree, click the + sign next to `MedRecEAR`, then right-click on `sessionEjbs`.

14. From the menu, select Define Policies and Roles for Individual Beans.

    A table listing all the EJBs that are in the JAR file appears.

    Note: Selecting this option allows you to create a scoped role for a particular EJB within an EJB JAR.

15. Click the [Define Scoped Roles] link for `AdminSessionEJB`.

16. Repeat steps 4 - 12 to create the scoped role named `MedRecSessionEJBAdmin` and grant this scoped role to the `MedRecAdmins` group.

Step 3: Secure the SessionEJB JAR.

1. In the navigation tree, right-click `sessionEjbs`.
2. From the menu, select Define Security Policy to display the Policy Editor page.

   Selecting this option indicates that you are creating a security policy at the EJB JAR level, which includes all EJBs within the JAR, and all methods within those EJBs.

3. In the Policy Condition list box, highlight `Caller is Granted the Role`.
4. Click Add to open the Roles window.
5. In the Enter Role Name field, type `MedRecSessionEJBPatient`.
6. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      MedRecSessionEJBPatient
   ```

   By defining this security policy for the `SessionEJB` JAR, you are overriding any security policies that have already been defined for the EJB resource type. If you completed "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console," you are overriding the inherited policy statement:

   ```
   Caller is Granted the Role
      MedRecAdmin
   ```

   that is shown in the Inherited Policy Statements list box. Otherwise, you will be overriding the default security policy:

```
Caller is a Member of the Group
   Everyone
```

For more information about default security policies, see Default Security Policies in *Securing WebLogic Resources*.

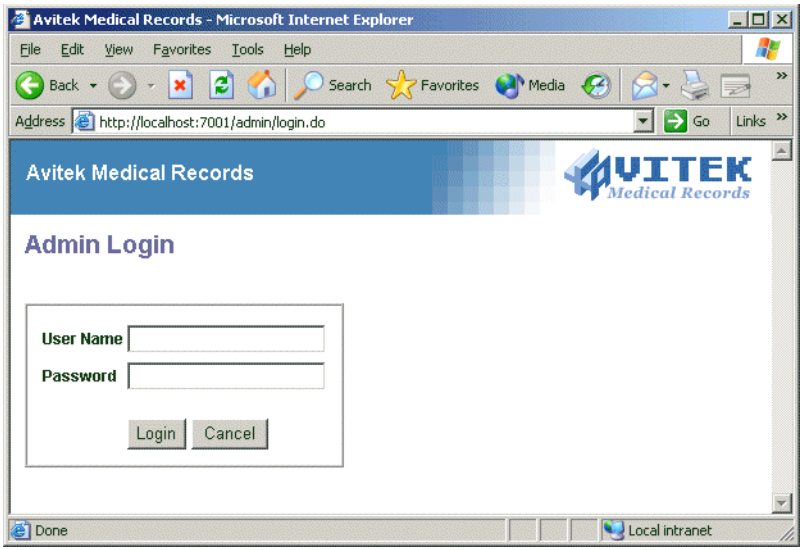
7. Click Apply to save your changes.

Step 4: Attempt to access an EJB in the SessionEJB JAR.

1. Open a new Web browser and type `http://localhost:7101/admin/login.do`.

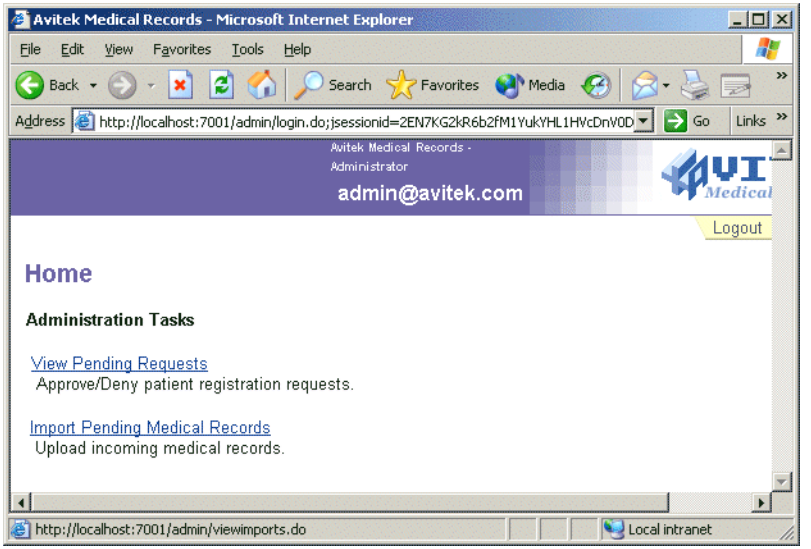
   The browser displays the login page shown in Figure 1.

Figure 1: Admin Login Page



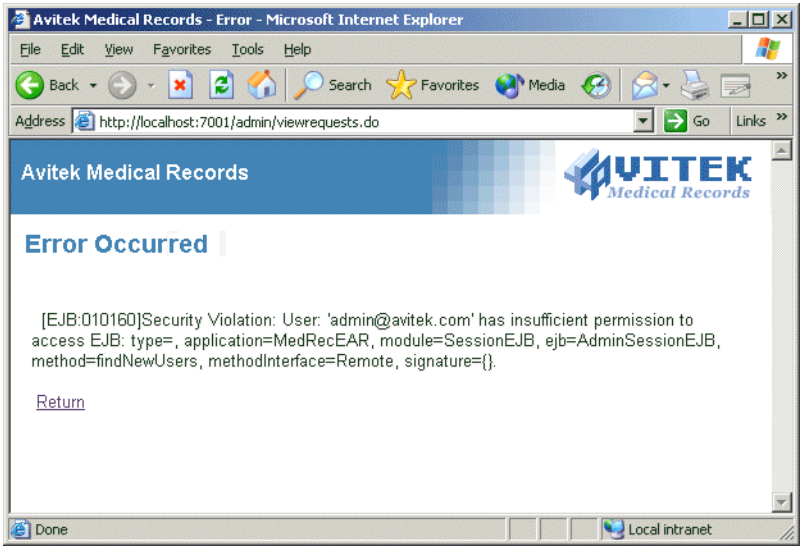
2. In the username field, type `admin@avitek.com`, and in the password field, type `weblogic`, then click Login.

Figure 2: Administration Tasks Page



3. On the Administration Tasks page shown in Figure 2, click the View Pending Requests link.

Figure 3: Error Page



The error page shown in Figure 3 is displayed because access to the `findNewUsers()` method in `AdminSessionEJB`, an EJB within the `SessionEJB` JAR you previously secured, is needed to view pending requests. User `admin@avitek.com` is not granted the `MedRecSessionEJBPatient` scoped role that was used to create the security policy, and is therefore is not granted access.

Step 5: Secure the AdminSessionEJB.

1. In the navigation tree at the left side of the Administration Console, right-click `sessionEjbs`.
2. From the menu, select Define Policies and Roles for Individual Beans.

   A table listing all the EJBs that are in the JAR file appears.

   Selecting this option enables you to create a security policy at the EJB level (meaning the security policy will apply to all methods within the EJB), or a particular method within the EJB.
3. Click the [Define Security Policies] link for `AdminSessionEJB` to display the Policy Editor page.
4. In the Policy Condition list box, highlight `Caller is Granted the Role`.

   Note: Do *not* modify the value shown in the Methods drop-down menu. (It should read: `ALL`.)
5. Click Add to open the Roles window.
6. In the Enter Role Name field, type `MedRecSessionEJBAdmin`.
7. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      MedRecSessionEJBAdmin
   ```

   By defining this security policy for `AdminSessionEJB`, you are overriding the security policy that has already been defined for the EJB JAR in Step 3: Secure the SessionEJB JAR. Specifically, you are overriding the inherited policy statement of:

   ```
   Caller is Granted the Role
   ```

`MedRecSessionEJBPatient`

that is shown in the Inherited Policy Statements list box.

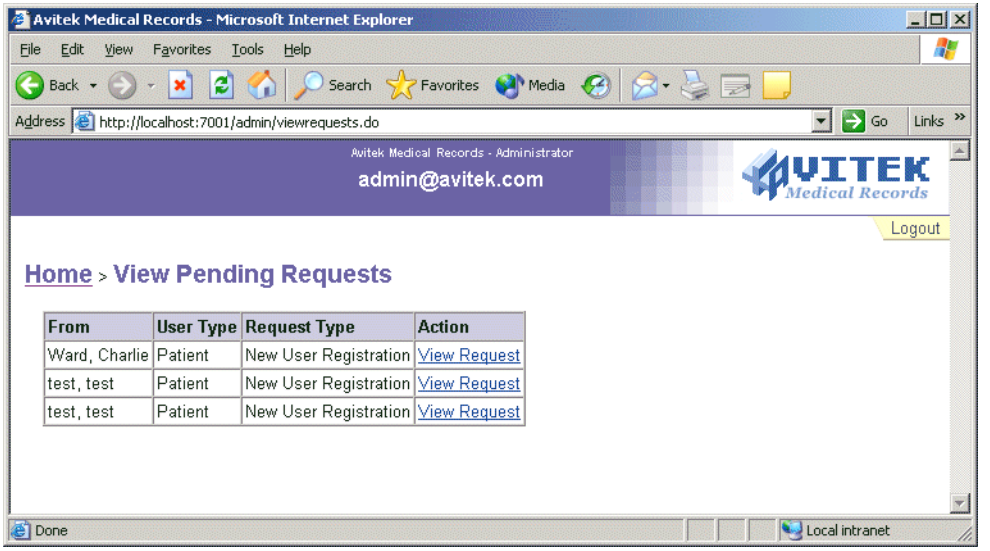
8. Click Apply to save your changes.

Step 6: Attempt to access AdminSessionEJB.

Repeat steps 1 - 3 in "Step 4: Attempt to access an EJB in the SessionEJB JAR." on page 1-6.

Instead of displaying the error page for step 3, the browser displays the View Pending Requests page shown in Figure 4.

Figure 4: View Pending Requests



This result occurs because user `admin@avitek.com` is granted the `MedRecEJBSessionAdmin` scoped role. This scoped role was used to create the security policy for `AdminSessionEJB`, the EJB containing the `findNewUsers()` method that is needed to view pending requests.

Step 7: Secure the findNewUsers() EJB method.

1. In the navigation tree at the left side of the Administration Console, right-click `sessionEjbs`.
2. From the menu, select Define Policies and Roles for Individual Beans.

   A table listing all the EJBs that are in the JAR file appears.

   Selecting this option enables you to create a security policy at the EJB level (meaning the security policy will apply to all methods within the EJB), or for a particular method within the EJB.
3. Click the [Define Security Policies] link for `AdminSessionEJB` to display the Policy Editor page.
4. Using the Methods drop-down menu, select the `findNewUsers() - REMOTE` method.
5. In the Policy Condition list box, highlight `Caller is Granted the Role`.
6. Click Add to open the Roles window.
7. In the Enter Role Name field, type `MedRecSessionEJBPatient`.

   You defined this scoped role on `SessionEJB`, but because the `findNewUsers()` method is a component of `AdminSessionEJB` (itself a component of `SessionEJB`), you can also use it here.
8. Click Add, then click OK.

   The Roles window closes. The Policy Statement list box reads:

   ```
   Caller is Granted the Role
      MedRecSessionEJBPatient
   ```

   By defining this security policy on the `findNewUsers()` method, you are overriding the security policy that has already been defined for `AdminSessionEJB` in "Step 5: Secure the AdminSessionEJB.." Specifically, you are overriding the inherited policy statement of:

   ```
   Caller is Granted the Role
      MedRecSessionEJBAdmin
   ```

   that is shown in the Policy Statement list box when `ALL` is selected from the Methods drop-down menu.

9. Click Apply to save your changes.

Step 8: Attempt to access the findNewUsers() EJB method.

Repeat steps 1 - 3 in "Step 4: Attempt to access an EJB in the SessionEJB JAR." on page 1-6.

The browser displays the error page shown in Figure 3. This result occurs because only users granted the scoped role `MedRecSessionEJBPatient` can access the `findNewUsers()` method, which is needed to view pending requests. User `admin@avitek.com` is not granted the scoped role that was used to create the security policy, and therefore is not granted access.

Best Practices

- The security realm settings are extremely important. If you want to secure URL (Web) resources using the WebLogic Server Administration Console rather than deployment descriptors, you must use the Check Roles and Policies/On Future Redeploys combination specified in "Step 1: Specify security realm settings.."
- If you have deployed an application (or module) with the On Future Redeploys drop-down menu set to Ignore Roles and Policies From DD one or more times *before* setting it to Initialize Roles and Policies From DD, you can still set security policies and security roles using the Administration Console. These changes will override any security specified in deployment descriptors.
- Do not use blank spaces, commas, hyphens, or any characters in this comma-separated list for user, group, or security role names: \t, < >, #, |, &, ~, ?, ( ), { }. User, group, and security role names are case sensitive. The proper syntax for a security role name is as defined for an `Nmtoken` in the Extensible Markup Language (XML) recommendation. The BEA convention is that group names are plural, and security role names are singular.
- It is inadvisable to create global roles and scoped roles with the same name. However, if you have a valid reason for doing this, know that the scoped role will override the global role if used in a `Caller is Granted the Role` policy condition.

- Scoped roles can be used in security policies from the level in the hierarchy where they are defined and below.
- BEA recommends assigning users to groups, then creating role statements using the `Caller is a Member of the Group` role condition. Individual users could also be granted a security role, but this is a less typical practice.
- BEA recommends using security roles (rather than users or groups) to secure WebLogic resources. Following this process makes it more efficient for administrators who work with large numbers of users.
- Create policy statements based on your organization's established business procedures.
- When creating new security policies, look for policy statements in the Inherited Policy Statement box of the Policy Editor page. If inherited policy statements exist, you will be overriding them.
- Remember that more-specific security policies override less-specific security policies. For example, a security policy on an EJB method overrides a security policy on the same EJB. Take care when overriding with less restrictive security policies (that is, giving a wider set of users access to a smaller set of components or WebLogic resources).
- You can delete all security settings for an application (or module) by deleting it entirely from the WebLogic Server domain and then redeploying it.

The Big Picture

This tutorial shows you how to secure application and various Enterprise JavaBean (EJB) resources using some examples. These examples may or may not be different from those used in the full MedRec application. However, the full MedRec application uses these same principles (as well as programmatic security) to secure EJB resources for both MedRec administrators and patients.

Related Reading

- *Securing WebLogic Resources*

- "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console" on page 1-1
- "Tutorial 19: Copying and Reinitializing Security Configurations" on page 1-1

1 Moving to Production Mode

Tutorial 19: Copying and Reinitializing Security Configurations

This tutorial describes how to copy a security configuration from deployment descriptors into the configured Authorization and Role Mapping providers' databases, so that you can use the Administration Console for subsequent modifications to security policies and security roles. It also describes how to reinitialize a security configuration using the original deployment descriptors.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create `MedRecDomain` and `MedRecServer`, and start `MedRecServer`. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development" on page -1.
- Obtain the "AdminWebApp Web Application" (available under Code Samples for WebLogic Server 8.1 on the *dev2dev Web site*), and unzip the `adminWebApp.zip` file to a temporary directory (for example, `C:\adminWebApp`).
- If you deployed the Enterprise Application named `MedRecEAR` as part of any prior tutorial, use the Administration Console to delete it.
- Read Using the Combined Technique to Secure Your URL (Web) and Enterprise JavaBean (EJB) Resources in *Securing WebLogic Resources*.

Procedure

This tutorial consists of three main steps:

Step 1: Copy a security configuration.

To copy security configurations for the `adminWebApp` Web application from its deployment descriptors into the configured Authorization and Role Mapping providers' databases, follow these steps:

Step 1: Specify security realm settings and deploy the Web application.

1. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
2. Click the `myrealm` security realm.
3. On the General tab, select All Web Applications and EJBs as the value for the Check Roles and Policies drop-down menu.

   This setting causes the WebLogic Security Service to perform security checks on *all* URL (Web) and EJB resources. For more information, see Understanding How to Check Security Roles and Security Policies in *Securing WebLogic Resources*.

   If All Web Applications and EJBs was already selected as the value of the Check Roles and Policies drop-down menu, just continue to step 4.
4. Select Initialize Roles and Polices From DD from the On Future Redeploys drop-down menu.

   This setting causes WebLogic Server to *copy* security configurations for URL (Web) and EJB resources from deployment descriptors into the configured Authorization and Role Mapping providers' databases *each time* you deploy the resource. For more information, see Understanding What to Do on Future Redeploys of the WebLogic Resource in *Securing WebLogic Resources*.
5. Click Apply to save your changes.
6. If you had to set the Check Roles and Policies drop-down menu to All Web Applications and EJBs in step 2 (that is, it was *not* already set this way), restart the server. (For help, see "Starting and Stopping WebLogic Servers: Quick Reference" in the *WebLogic Server Administration Guide*.)

   If you did not have to modify the value of the Check Role and Policies drop-down menu in step 3, continue to step 7 *without restarting the server.*
7. Deploy the `adminWebApp` Web Application module and target it to the `MedRecServer`.

   For instructions about how to deploy Web Applications, see *Deploying WebLogic Server Applications*.

Step 2: Verify the copied security policies (optional).

1. Open the `web.xml` deployment descriptor for the `adminWebApp` Web application, and record the content of any `<url-pattern>` and `<http-method>` elements, as well as any `<role-name>` subelements of the `<auth-constraint>` element. Listing 1 shows the relevant portions of the `web.xml` deployment descriptor file in bold font.

Listing 1: The adminWebApp Web Application web.xml Deployment Descriptor

```
<!DOCTYPE web-app (View Source for full doctype...)>
<web-app>
   ...
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>images</web-resource-name>
         <url-pattern>*.gif</url-pattern>
      </web-resource-collection>
   </security-constraint>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>UnsecureLoginAction
         </web-resource-name>
         <url-pattern>login.do</url-pattern>
      </web-resource-collection>
   </security-constraint>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>UnsecureLoginPages
         </web-resource-name>
         <url-pattern>Login.jsp</url-pattern>
      </web-resource-collection>
   </security-constraint>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>AdminActions</web-resource-name>
         <description>These pages are only accessible by authorized
administrators.</description>
         <url-pattern>*.do</url-pattern>
         <url-pattern>*.jsp</url-pattern>
         <http-method>POST</http-method>
         <http-method>GET</http-method>
      </web-resource-collection>
      <auth-constraint>
```

```
        <description>These are the roles who have access.</description>
        <role-name>admin</role-name>
      </auth-constraint>
      ...
   </security-constraint>
   ...
   <security-role>
      <description>An administrator</description>
      <role-name>admin</role-name>
   </security-role>
</web-app>
```

2. In the navigation tree at the left side of the Administration Console, expand Web Application Modules, then right-click `adminWebApp`.
3. From the menu, select Define Security Policy to display the General tab.

   There are five hyperlinked URL patterns that correspond to those you recorded in step 1 listed under the Already Defined URL Patterns heading.
4. Click the hyperlinked URL pattern `*.do` to display the Policy Editor page.
5. Using the Methods drop-down menu, select `POST`.

   The `Caller is Granted the Role` Policy Condition is highlighted and the Policy Statement list box reads:

   ```
   Caller is Granted the Role

      admin
   ```

   If you click a hyperlinked URL pattern that did not have a corresponding `<http-method>` element in the `web.xml` deployment descriptor, the Policy Statement list box displays the appropriate security policy when the Methods drop-down menu contains the value `ALL`. For example, the security policy for the URL pattern `*.gif` from Listing 1 can be viewed when the Methods drop-down reads `ALL`.

   If the URL pattern does not have a corresponding `<auth-constraint>` element in the `web.xml` deployment descriptor, the security policy for that URL pattern will be created using the `Anonymous` global role (for example, the security policy for the URL pattern `*.gif` from Listing 1). For more information about default global roles, see Default Global Roles in *Securing WebLogic Resources*.
6. Repeat steps 2 - 5 to verify multiple security policies.

Step 3: Verify the copied security roles (optional).

1. Open the `weblogic.xml` deployment descriptor for the `adminWebApp` Web Application, and record the content of any `<security-role-assignment>` elements, specifically focusing on the `<role-name>` and `<principal-name>` subelements. Listing 2 shows the relevant portions of the `weblogic.xml` deployment descriptor file in bold font.

Listing 2: The adminWebApp Web Application weblogic.xml Deployment Descriptor

```
<!DOCTYPE weblogic-web-app (View Source for full doctype...)>
   <weblogic-web-app>
      <context-root>admin</context-root>
      <security-role-assignment>
         <role-name>admin</role-name>
         <principal-name>admin</principal-name>
      </security-role-assignment>
   </weblogic-web-app>
```

2. In the navigation tree at the left side of the Administration Console, right-click on the `adminWebApp` Web Application.
3. From the menu, select Define Scoped Role to display the General tab.
4. Click the hyperlinked URL pattern `/*`.

   The Scoped Roles page displays all the scoped roles for this Web Application that are currently defined in the WebLogic Role Mapping provider's database, including the scoped role called `admin`.

   Security roles obtained from deployment descriptors are always copied into the configured Role Mapping provider's database as scoped roles, with a URL pattern of `/*`.
5. Click the hyperlinked scoped role `admin`.
6. Select the Conditions tab.

   The Role Statement list box contains a Role Statement based on the content of the deployment descriptor's corresponding `<principal-name>` element, which in this case is a user or group called `admin`.

Step 4: Revert the On Future Redeploys setting.

Caution: You must perform this step. Failure to revert this setting may result in inconsistent security configurations when your URL (Web) resources are redeployed. If you do not perform this step or perform this step incorrectly, you see the following message the next time you load the Policy Editor page:

```
The information presented below may not be accurate. To
ensure that you are viewing accurate information, you may
need to delete and redeploy your WebLogic resources.
```

1. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
2. Click the `myrealm` security realm.
3. On the General tab, select Ignore Roles and Polices From DD as the value for the On Future Redeploys drop-down menu.

   This setting indicates that you will set security for URL (Web) and EJB resources using the Administration Console, not deployment descriptors. For more information, see Understanding What to Do on Future Deploys of the WebLogic Resource in *Securing WebLogic Resources*.
4. Click Apply to save your changes.

Step 2: Modify a security policy using the Administration Console.

1. In the navigation tree at the left side of the Administration Console, expand Web Application Modules, then right-click `adminWebApp`.
2. From the menu, select Define Security Policy to display the General tab.

   Five hyperlinked URL patterns correspond to those you recorded in "Step 2: Verify the copied security policies (optional)." on page 1-4 listed under the Already Defined URL Patterns heading.
3. Click the hyperlinked URL pattern `*.do` to display the Policy Editor page.
4. Using the Methods drop-down menu, select `POST`.

   The `Caller is Granted the Role` Policy Condition is highlighted and the Policy Statement list box reads:

```
Caller is Granted the Role
   admin
```

5. In the Policy Condition list box, highlight the `Hours of Access are Between` policy condition.
6. Click Add, then click OK in the Time Constraint window to select the default start and end times.

   The Policy Statement list box reads as follows:

```
Caller is Granted the Role
   developers
and Hours of Access are Between
   08:00:00 and 19:00:00
```

7. Click Apply to save your changes.

Step 3: Reinitialize a security configuration.

To reinitialize security configurations for the `adminWebApp` Web Application from its deployment descriptors, follow these steps:

- "Step 1: Modify the On Future Redeploys setting." on page 1-8
- "Step 2: Redeploy the adminWebApp Web application." on page 1-9
- "Step 3: Verify that the security configuration has been reinitialized (optional)." on page 1-9
- "Step 4: Revert the On Future Redeploys setting." on page 1-10

Step 1: Modify the On Future Redeploys setting.

1. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
2. Click the `myrealm` security realm.
3. On the General tab, from the On Future Redeploys drop-down menu, select Initialize Roles and Polices From DD.

   This setting means that WebLogic Server will *copy* security configurations for URL (Web) and EJB resources from deployment descriptors into the configured

Authorization and Role Mapping providers' databases *each time* you deploy the resource. For more information, see Understanding What to Do on Future Redeploys of the WebLogic Resource in *Securing WebLogic Resources*.

If All Web Applications and EJBs was already selected as the value of the Check Roles and Policies drop-down menu, just continue to step 4.

4. Click Apply to save your changes.

Step 2: Redeploy the adminWebApp Web application.

1. In the navigation tree at the left side of the Administration Console, expand Deployments->Web Application Modules.
2. Click the `adminWebApp` Web application.

   A table that lists all the Web application or EJB modules appears in the right pane.
3. Click the trash can icon that is located in the same row as the `adminWebApp` Web Application.
4. Click Yes, then the Continue link to delete the `adminWebApp` Web Application.

   The `adminWebApp` Web Application no longer appears in the table.
5. Click the Deploy button that corresponds to `MedRecServer`, to which you targeted the `adminWebApp` Web Application module.
6. Re-deploy the `adminWebApp` Web Application, targeting it to `MedRecServer`.

   Note: For instructions about how to deploy Web Application and EJB modules, see *Deploying WebLogic Server Applications*.

Step 3: Verify that the security configuration has been reinitialized (optional).

1. In the navigation tree at the left side of the Administration Console, right-click `adminWebApp`.
2. From the menu, select Define Security Policy to display the General tab.

   Five hyperlinked URL patterns correspond to those you recorded in step 1 listed under the Already Defined URL Patterns heading.
3. Click the hyperlinked URL pattern `*.do` to display the Policy Editor page.

4. Using the Methods drop-down menu, select `POST`.

   The `Caller is Granted the Role` Policy Condition is highlighted and the Policy Statement list box reads:

   ```
   Caller is Granted the Role

      admin
   ```

   The policy statement you created using the `Hours of Access are Between` policy condition in "Step 2: Modify a security policy using the Administration Console." on page 1-7 is gone, because it was not defined in the deployment descriptor from which you just initialized the security configuration.

Step 4: Revert the On Future Redeploys setting.

Caution: You must perform this step. Failure to revert this setting may result in inconsistent security configurations when your URL (Web) resources are redeployed. If you do not perform this step or perform this step incorrectly, you see the following message the next time you load the Policy Editor page:

```
The information presented below may not be accurate. To
ensure that you are viewing accurate information, you may
need to delete and redeploy your WebLogic resources.
```

1. In the navigation tree at the left side of the Administration Console, expand Security->Realms.
2. Click the `myrealm` security realm.
3. On the General tab, select Ignore Roles and Polices From DD as the value for the On Future Redeploys drop-down menu.

   Note: This setting means that you will set security for URL (Web) and EJB resources using the Administration Console, not deployment descriptors. For more information, see Understanding What to Do on Future Deploys of the WebLogic Resource in *Securing WebLogic Resources*.

4. Click Apply to save your changes.

Best Practices

- Do not use blank spaces, commas, hyphens, or any characters in this comma-separated list for user, group, or security role names: \t, < >, #, |, &, ~, ?, ( ), { }. User, group, and security role names are case sensitive. The proper syntax for a security role name is as defined for an `Nmtoken` in the Extensible Markup Language (XML) recommendation. The BEA convention is that group names are plural, and security role names are singular.
- Remember that redeploying a WebLogic resource with the On Future Redeploys drop-down menu set to Ignore Roles and Policies From DD *does not* affect the security configuration (that is, security policies or security roles) of the resource.
- When the On Future Redeploys drop-down menu is set to Initialize Roles and Policies From DD, *any* redeploy of a WebLogic resource will update the security configuration. This includes targeting a new server and setting a server with an application or module targeted to it to bounce. Take care when altering security policies and security roles that are specified in a deployment descriptor when the On Future Redeploys drop-down menu is set to Initialize Roles and Policies From DD.
- If you initialize a security configuration from deployment descriptors and then customize security policies and security roles using the Administration Console, make sure that you *never* boot a server when the value of the On Future Redeploys drop-down menu is Reinitialize Roles and Policies From DD. If you do, then all the security policy and security role customizations you performed using the Administration Console for all of your Web applications (and EJBs) will be lost.
- Always have the On Future Redeploys set to Ignore Roles and Policies From DD *except* when:
  - You are about to deploy a new Web Application or EJB module
  - You want to redeploy a Web Application or EJB module and initialize its security configuration (security policies and security roles).

The Big Picture

This tutorial shows you how to copy the security configuration for a Web Application from its deployment descriptors into the configured Authorization and Role Mapping providers' databases, so that you can use the Administration Console for subsequent modifications to the Web Application's security roles and security policies. The same example shows you how to reinitialize the security configuration using the Web Application's original deployment descriptors.

The full MedRec application uses the principles described in "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console" on page 1-1 and "Tutorial 18: Securing Enterprise JavaBean (EJB) Resources Using the Administration Console" on page 1-1 (as well as programmatic security) to secure EJB resources for both MedRec administrators and patients.

Related Reading

- *Securing WebLogic Resources*
- "Tutorial 17: Securing Application and URL (Web) Resources Using the Administration Console" on page 1-1
- "Tutorial 18: Securing Enterprise JavaBean (EJB) Resources Using the Administration Console" on page 1-1

1 Moving to Production Mode

Tutorial 20: Redeploying the MedRec Package

This tutorial shows how to use the Administration Console to redeploy the MedRec application to MedRecServer in a production environment. The MedRec applications are contained in the `dist` directory, packaged in three directories in the recommended exploded format.

Redeploy an application if you have updated its class files or its generated deployment descriptor files.

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Work through Tutorial 13: Packaging MedRec for Distribution.
- Work through Tutorial 14: Deploying the MedRec Package for Production, and have the package currently deployed to `MedRecServer`.

Procedures

This demonstration includes three separate procedures:

- Procedure 1: Edit a deployment descriptor without redeploying.
- Procedure 2: Refresh a static file without redeploying the application.
- Procedure 3: Redeploy the entire application.

Procedure 1: Edit a deployment descriptor without redeploying.

Use the Administration Console to modify certain deployment descriptor elements and their attributes for Applications that are deployed as exploded archive files. You cannot edit these descriptors for applications packaged as EARs.

In this procedure, change the value that determines the number of seconds a Web Application remains idle before timing out.

1. Open the Administration Console.

   Browse to `http://localhost:7101/console`, where `localhost` is the network name of your computer.

2. In the left pane of the Console, expand Deployment and select Applications.

   The Applications table displays all deployed applications, which include the `medrecEar`, `physicianEar`, and `startupEar` applications you deployed in Tutorial 15: Deploying the MedRec Package for Production.

3. In the right panel, expand `medrecEAR` and select `patient` to select the `patient` Web Application.

4. In the right pane, select Configuration->Descriptor.

   The descriptor elements displayed in the Descriptor tab are limited to descriptor elements that can be dynamically changed at runtime.

5. Scroll down to the Deployment Descriptors box and click `weblogic.xml` to open it in a separate window.

6. Locate the `session-descriptor` element, which should look like this stanza:

```
<session-descriptor>
  <session-param>
    <param-name>TimeoutSecs</param-name>
    <param-value>600</param-value>
  </session-param>
```

7. Return to the WebLogic Server Administration Console and edit the TimeoutSecs parameter by adding a "1" before the "600."

8. Click Apply.

9. Return to the `weblogic.xml` page and refresh your browser to see the updated `param-value`, which is now in effect for the application.

Procedure 2: Refresh a static file without redeploying the application.

Use the `weblogic.Deployer` utility to notify the server when static files have changed.

In this procedure, you change an image, refresh the image file on the server, and view the refreshed file in the Web Application. Clean up the application by restoring the image and refreshing the file again.

Use the file `logo.gif` in the `physicianWebApp` component of `physicianEar`. The Web Application references this file from a virtual directory specified in the `weblogic.xml` file located in the `WEB-INF` directory (not from the Web Application images directory).

The relevant stanza from `weblogic.xml` follows:

```
<virtual-directory-mapping>
```

```
    <local-path>C:/bea/weblogic81sp1/weblogic81/samples/server/m
    edrec/src/common/web</local-path>

    <url-pattern>images/*</url-pattern>

  </virtual-directory-mapping>
```

1. Save `logo.gif` to an alternate name such as `logo1.gif`.
2. Save a different GIF file to `logo.gif`.
3. Open a command window and set your environment

```
WL_HOME\samples\domains\medrec> setMedRecEnv.cmd
```

Change to the application directory,
`WL_HOME\samples\server\medrec\dist\physicianEar`.

4. Enter the redeploy command, specifying logo.gif.

```
java weblogic.Deployer -adminurl http://localhost:7101 -user
weblogic -password weblogic -name physicianEar -redeploy
..\..\src\common\web\images\logo.gif
```

The server reports on the task:

```
Initiated Task: [11] [Deployer:149026]Redeploy application
physicianEar on MedRecServer.

Task 11 completed: [Deployer:149026]Redeploy application
physicianEar on MedRecServer.

Deployment completed on Server MedRecServer
```

5. Rename `logo1.gif` to `logo.gif`.
6. Repeat the redeploy command:

```
java weblogic.Deployer -adminurl http://localhost:7101 -user
weblogic -password weblogic -name physicianEar -redeploy
..\..\src\common\web\images\logo.gif
```

Procedure 3: Redeploy the entire application.

In this procedure it is assumed that MedRec is deployed to a currently running instance of MedRecServer. Follow these steps to update a deployed application whose class files or generated deployment descriptor files have been changed.

1. Open the Administration Console.

Browse to `http://localhost:7101/console`, where `localhost` is the network name of your computer.

2. In the left pane of the Console, expand Deployments and select Applications.

   The Applications table displays all deployed applications, which include the `medrecEAR`, `physicianEAR`, and `startupEAR` applications you deployed in Tutorial 15: Deploying the MedRec Package for Production.

3. Redeploy all three applications, starting with `medrecEAR`.

   a. Click on `medrecEAR`.

      In the right-hand panel, the `medrecEAR` Configuration tab displays configuration details.

   b. Select the Deploy tab.

      The Deploy panel lists deployment status of EJB modules and Web Application modules.

   c. Click Redeploy Application.

   d. Return to the Deploy->Applications panel, select `medrecEAR`, and repeat steps b and c.

   e. Return to the Deploy->Applications panel, select `startupEAR`, and repeat steps b and c.

Best Practices

Redeploying an application in production is a serious undertaking that can affect performance, so plan application updates carefully. Redeploying an application re-sends the entire application over the network to all of the servers targeted by that Web Application. Increased network traffic may affect network performance when an application is re-sent to the Managed Servers. If the application is currently in production and in use, redeploying causes WebLogic Server to lose all active HTTP sessions.

If you have only modified static files, it is probably possible to refresh the files without redeploying the entire application. See Redeploying Static Files in a Web Application in *Deploying WebLogic Server Applications*.

Some deployment descriptor elements can be modified without redeploying the application. See Viewing and Updating Deployment Descriptors in the *Administration Console Online Help*.

The Big Picture

This tutorial explains how to redeploy an application in production using the Administration Console. You can also use the command-line `weblogic.Deploy` tool to redeploy applications, and to refresh static files in a deployed application.

If you have added modules in your application, redeploying the application deploys the current modules. If you have deleted modules from your application, explicitly remove them from the application domain to remove them from deployment. See Removing an Application or Module from the Domain in *Deploying Applications and Modules*.

Related Reading

- Deploying Applications and Modules