BEA WebLogic Server®

Avitek Medical Records Development Tutorials

Version 9.1
Revised: December 20, 2005

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Moving to Production

CHAPTER 1

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 2.0) as a basis for instruction. However, you can easily apply the procedures and best practices to your own J2EE applications.

The following sections provide an overview of the application and the tutorials:

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 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 log-in 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, medrecEar and physicianEar, and two supporting applications, startBrowserEar and initEar, that load the MedRec informational page and register management beans (MBeans). The main applications support one or more user scenarios for MedRec:

- `medrecEar`—Patients log in to the patient Web Application (`patient`) 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 (`admin`) 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.

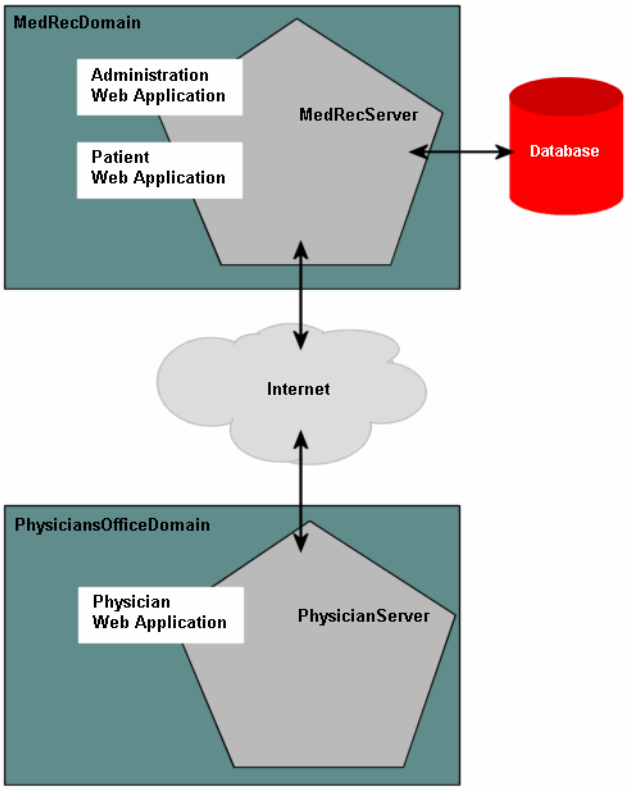
- `physicianEar`—Physicians and nurses log in to the physician Web Application (`physician`) to search and access patient profiles, create and review patient medical records, and prescribe medicine to patients. The physician application communicates through the Web Service provided in `medrecEar`.

- `startBrowserEar`—The `startBrowserEar` 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.

- `initEar`—The initEar application implements and registers a custom management bean (MBean) that polls the database every 6 seconds to check for new users to be added to the system.

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 1-1 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.

Although the MedRec tutorials explain how to develop application components using WebLogic Server tools, they do not describe the MedRec J2EE implementation or explain how to program J2EE components in Java.

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 provides 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.

Because it is assumed in these tutorials that you are working on a Microsoft Windows computer, all commands use Windows syntax.

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 (such as persistent stores, JMS queues, and store-and-forward agents) required to deploy the MedRec application.
- Developing 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 to generate deployment descriptors, package, and deploy J2EE components, as well as program some components, such as Web Services.
- Moving to Production describes how to take the MedRec application from the development environment into a production environment. Tutorials in this section focus on packaging, deploying, and securing the MedRec application.

Related Reading

- Introduction to WebLogic Server and WebLogic Express
- Developing Applications on BEA WebLogic Server (topic page)
- Developing Applications with WebLogic Server (Programming Guide)

CHAPTER 2

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 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, along with other supporting configuration XML files, 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:

Prerequisites

Before starting this tutorial:

- Make sure WebLogic Server 9.1 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: Enable log4j for logging application messages.
- Step 3: Add the MedRecDBMSPlugin.jar to the server's CLASSPATH.
- Step 4: Start the MedRec Administration Server.

Step 1: Create the MedRec domain and MedRec server.

The MedRec domain includes a single server that will host the MedRec back-end services, the MedRec Administration application, and the Patient application (both applications are Web applications).

1. Launch the Configuration Wizard:

   Start→All Programs→BEA Products→Tools→Configuration Wizard

2. In the Welcome window, select **Create a new WebLogic domain**.
3. Click Next.
4. In the Select Domain Source window, select the option that begins **Generate a domain configured automatically ...**

   For the sake of simplicity, do not check AquaLogic Service Bus or Apache Beehive.

5. Click Next.
6. In the Configure Administrative Username and Password window, enter:
   - `weblogic` for User Name

- `weblogic` for User Password and Confirm user Password
- An optional description.

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

Note: In a production environment the user name and password should not be the same.

7. Click Next.
8. In the Configure Server Start Mode and JDK window, select:
   - `Development Mode` for WebLogic Domain Startup Mode
   - `Sun SDK 1.5.0_XX @ JDK_location`, under the BEA Supplied JDKs option, for JDK 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.
9. Click Next.
10. In the Customize Environments and Services Settings window, select Yes.
11. Click Next.
12. In the Configure the Administration Server window, enter or select:
    - `MedRecServer` for Name.
    - `All Local Addresses` for Listen Address.
    - `7101` for Listen Port.

      This tutorial specifies that you enter a port number that is deliberately different from the default port number of the Examples server (`7011`), which is also the general default WebLogic Server port number. If you have already configured this number for another WebLogic Server instance, enter a different value to avoid network communication conflicts. 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.

      This tutorial specifies that you enter an SSL port number that is deliberately different from the default SSL port number (`7012`). If you have already configured this number

for the SSL listen port of another WebLogic Server instance, enter a different value to avoid network communication conflicts. 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.

13. Click Next.
14. Select Next in the following windows without making any changes:
    - Configure Managed Servers
    - Configure Machines
15. In the Review WebLogic Domain window, click on `MedRecServer` in the left pane and review your choices, then click Next.
16. In the Create WebLogic Domain window:
    a. Enter `MedRecDomain` as the Domain Name.
    b. Click Create to create the MedRec domain in the folder displayed in the Domain Location text box. When the Configuration Wizard finishes creating the domain, the `Domain Created Successfully!` message is displayed.
17. Click Done to close the Configuration Wizard.

    Do not check Start Admin Server.

Step 2: Enable log4j for logging application messages.

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. You also need to copy the log4j jar files to the lib directory of the MedRec domain. To complete these steps:

1. Copy the log4j properties file from the pre-configured MedRec domain (under the main WebLogic Server installation directory) to the new domain you just created.

   For example, if you installed WebLogic Server in the `c:\bea` directory, then enter the following in a command-line shell:

   ```
   prompt> copy c:\bea\weblogic91\samples\domains\medrec\log4jConfig.xml
   c:\bea\user_projects\domains\MedRecDomain
   ```

2. Open the `setDomainEnv.cmd` script, located in the `bin` directory of your newly created domain directory, for your new domain in a text editor. For example:

```
prompt> notepad
c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
```

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

   ```
   if NOT "%LOG4J_CONFIG_FILE%"=="" (
   ```

4. Add the following two lines immediately before the line specified in the preceding step:

   ```
   set
   LOG4J_CONFIG_FILE=c:\bea\user_projects\domains\MedRecDomain\log4jConfig
   .xml

   for %%i in ("%LOG4J_CONFIG_FILE%") do set LOG4J_CONFIG_FILE=%%~fsi
   ```

   It is assumed in the preceding text that the MedRecDomain directory is `c:\bea\user_projects\domains\MedRecDomain`; enter your exact domain directory if it is different.

5. Save the file and exit your text editor.

6. Copy the log4j JAR files from the `lib` directory of the pre-configured MedRec domain to the `lib` directory of the new domain you just created:

   ```
   prompt> copy c:\bea\weblogic91\samples\domains\medrec\lib\*.jar
   c:\bea\user_projects\domains\MedRecDomain\lib
   ```

   All JAR files in the `lib` subdirectory of a domain directory are automatically added to the WebLogic Server system CLASSPATH when the servers in the domain start up.

Step 3: Add the MedRecDBMSPlugin.jar to the server's CLASSPATH.

The MedRec application suite uses a custom DBMS authenticator to retrieve login credentials from the configured PointBase RDBMS for a given username. The Java class that implements this authenticator is in the `MedRecDBMSPlugin.jar` file, located in the already-installed MedRec domain. You must add the JAR file to the MedRecServer's CLASSPATH so that you can successfully configure the custom authenticator using the Administration Console (described in Tutorial 3: Configuring WebLogic Server Resources with the Administration Console).

1. Open the `setDomainEnv.cmd` script, located in the `bin` directory of your domain directory, for your new domain in a text editor. For example:

   ```
   prompt> notepad
   c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

2. Find the following line in the `setDomainEnv.cmd` script:

```
set PRE_CLASSPATH=
```

3. Update the `PRE_CLASSPATH` variable by adding the full path name of the `MedRecDBMSPlug.jar` file, which is located in the *`WL_HOME`*`\samples\domains\medrec\security` directory, where *`WL_HOME`* refers to the root directory of WebLogic Server (default `c:\bea\weblogic91`). For example:

```
set
PRE_CLASSPATH=c:\bea\weblogic91\samples\domains\medrec\security\MedRecD
BMSPlugin.jar
```

4. Save the file and exit your text editor.

Step 4: Start the MedRec Administration Server.

Start the MedRec Administration Server using one of the following methods:

From the Start menu:

```
Start→All Programs→BEA Products→User Projects→MedRecDomain→Start
Admin Server for WebLogic Server Domain
```

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 command prompt, enter:

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

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

   (Windows) `prompt> startWebLogic.cmd`

   (UNIX) `prompt> ./startWeblogic.sh`

During startup, you will see a message similar to the following once the server is ready to be used:

```
<Dec 9, 2005 10:52:20 AM PST> <Notice> <WebLogicServer> <BEA-000360>
<Server started in RUNNING mode>
```

Best Practices

- Use the Configuration Wizard to create and configure domains. The Configuration Wizard creates the necessary configuration file (`config.xml`), supporting configuration XML files, 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\weblogic91`.
- Use the `lib` subdirectory of your domain directory to add one or more JAR files to the WebLogic Server system CLASSPATH when servers start up. The `lib` subdirectory is intended for JAR files that change infrequently and are required by all or most applications deployed in the server, or by WebLogic Server itself. For example, you might use the `lib` directory to store third-party utility classes that are required by all deployments in a domain. You can also use it to apply patches to WebLogic Server.

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 WebLogic Domains Using the Configuration Wizard
- Starting and Stopping Servers: Quick Reference in *Managing Server Startup and Shutdown*
- Start and stop servers in the *Administration Console Online Help*
- Understanding WebLogic Server Application Classloading in *Developing Applications with WebLogic Server*
- Log4j and the Commons Logging API in *Configuring Log Files and Filtering Log Messages*
- Authentication Providers in *Developing Security Providers for WebLogic Server*

CHAPTER 3

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 describes how to:

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

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

Warning: The PointBase database management system shipped with WebLogic Server is meant to be used for evaluation purposes only. You cannot use it in production without a license from PointBase.

The tutorial includes the following sections:

- Prerequisites
- Procedure
- Best Practices
- The Big Picture

- Related Reading

Prerequisites

Create the MedRec domain and MedRec server instance. See "Tutorial 1: Creating a WebLogic Domain and Server Instance for Development".

Procedure

To start and use PointBase:

- Step 1: Start the PointBase database.
- Step 2: Use the PointBase console to view the MedRec tables and data.

Step 1: Start the PointBase database.

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

   ```
   prompt> cd c:\bea\weblogic91\common\eval\pointbase\tools
   ```

3. Start the PointBase database:

   ```
   prompt> startPointBase.cmd
   ```

   Once the PointBase database server starts, you see a message:

   ```
   Server started, listening on port 9092, display level: 3 ...
   ```

4. Leave this command window open to keep 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 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 Products—Examples—WebLogic Server—PointBase Console

From a script:

a. In a command-line shell, go to the *bea_home*\weblogic91\common\eval\pointbase\tools directory where *bea_home* is the main BEA home directory, typically c:\bea. For example:

```
prompt> cd c:\bea\weblogic91\common\eval\pointbase\tools
```

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

```
prompt> 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. Select Open Specified Database.
7. Click OK.
8. In the left pane, expand Schemas—MedRec.
9. Browse the tables that make up the MedRec database.
10. Click File—Exit to close the PointBase console.

Best Practices

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

  c:\bea\weblogic91\common\eval\pointbase\tools

- Automatically start dependent processes, such as the PointBase database, by updating the scripts that start WebLogic Server and adding calls to the relevant scripts, such as startPointBase.cmd, at the beginning of the WebLogic Server start script. Alternatively, create a master script to handle the control of starting all required processes for development, including a call to start WebLogic Server.

See Step 4: Start the MedRec Administration Server. in Tutorial 1 for information about the WebLogic Server start scripts.

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, the application that uses each EJB, and the PointBase tables in which the information is persisted.

Table 3-1 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	Patient information such as name, address reference to the ADDRESS table, SSN, and so on.
PhysicianEJB	Administration	PHYSICIAN	Physician information such as name, address reference to the ADDRESS table, phone, and email.

Table 3-1 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 (MDB) can process them. The messages are stored in the `MEDRECWLSTORE` PointBase table.

This table is generated automatically when you use the Administration Console to create the JMS JDBC store. The table is used internally by JMS.

Related Reading

- *PointBase Console Guide*
- *PointBase Developer's Guide*

- *PointBase System Guide*
- Understanding Enterprise Java Beans (EJB) in *Programming WebLogic Enterprise JavaBeans*
- Designing Enterprise Java Beans in *Programming WebLogic Enterprise JavaBeans*
- Entity EJBs in *Programming WebLogic Enterprise JavaBeans*

CHAPTER 4

Configuring Domains and Servers

Tutorial 3: Configuring WebLogic Server Resources with the Administration Console

This tutorial describes how to configure the WebLogic Server resources required to deploy and run the MedRec application on the MedRec server. These resources include:

- Java Database Connectivity (JDBC) global data sources for the PointBase database management system.
- Java Message Service (JMS) persistent stores, JMS servers, and queues.
- Store and Forward (SAF) agents to be used by Web Services reliable messaging
- JavaMail mail sessions
- Custom DBMS Authenticators

The tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create the MedRec domain and MedRec server, and start the MedRec administration 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 stand-alone JDBC data sources.
- Step 3: Create the file stores used by Web Services reliable messaging.
- Step 4: Create a JMS JDBC store.
- Step 5: Create JMS servers.
- Step 6: Create the stand-alone JMS module and queue.
- Step 7: Create the Store and Forward (SAF) agent.
- Step 8: Add email capabilities to the MedRec application.
- Step 9: Configure the MedRec Custom DBMS 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. Open the Administration Console by navigating in a browser to:

   `http://host:7101/console`

where *host* refers to the computer on which `MedRecServer` is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.

Step 2: Create the stand-alone JDBC data sources.

In WebLogic Server, you configure database connectivity by adding JDBC data sources to your WebLogic domain. A data source is a J2EE standard method of configuring connectivity to a database. A data source can be deployed by itself as a stand-alone module, or deployed as part of an Enterprise application as a packaged module.

Each WebLogic data source contains a pool of database connections. Applications look up the data source in the JNDI tree or in the local application context and then reserve a database connection with the `getConnection` method. Data sources and their connection pools provide connection management processes that help keep your system running and performing well.

This procedure describes how to create two stand-alone JDBC data sources: the first uses an XA JDBC driver and the second one does not. Typically you always use an XA JDBC driver when creating a data source. However, because JMS JDBC stores do not support XA resource drivers (WebLogic JMS implements its own XA resource), a second non-XA data source is needed. A later procedure shows how to associate the non-XA data source to a JMS JDBC store.

1. Click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services—JDBC.
3. Click Data Sources.
4. In the right pane, click New.
5. On the JDBC Data Source Properties page:
   - In the Name field, enter `MedRecGlobalDataSource`.
   - In the JNDI Name text box, enter `jdbc/MedRecGlobalDataSource`.
   - Select `PointBase` as the Database Type.
   - Select `PointBase's Driver (Type 4) Versions:4.X 5.X` as the Database Driver.
6. Click Next.

7. On the Transaction Options page, un-check Supports Global Transactions. Leave One-Phase Commit checked.

   This JDBC data source will be used to create a JDBC store, which requires that the data source *not* support global transactions.

8. Click Next.

9. On the Connection Properties page:

   – In the Database Name field, enter `demo`.

   – In the Host Name field, enter `localhost` (default value).

   – In the Port field, enter `9092` (default value).

   – In the Database User Name field, enter `medrec`.

   – In the Password and Confirm Password fields, enter `medrec`.

10. Click Next.

11. In the Test Database Connection page, ensure that the information to test the connection to the PointBase database is correct.

    Text field values are based on information you have already provided as well as default PointBase values (such as the Pointbase driver class name of `com.pointbase.jdbc.jdbcUniversalDriver`).

    Click Test Configuration to test the connection; the message `Connection test succeeded` appears in the Messages pane if you have configured the data source correctly.

    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.

12. Click Next.

13. Select the MedRecServer target.

14. Click Finish.

    The new data source is listed in the Data Sources table.

15. In the Summary of JDBC Data Sources page, click New.

16. On the JDBC Data Source Properties page:

    – In the Name field, enter `MedRecGlobalDataSourceXA`.

    – In the JNDI Name text box, enter `jdbc/MedRecGlobalDataSourceXA`.

   - Select `PointBase` as the Database Type.
   - Select `PointBase's Driver (Type 4XA) Versions:4.X 5.X` as the Database Driver.

17. Click Next.
18. On the Transaction Options page, click Next.
19. On the Connection Properties page:
    - In the Database Name field, enter `demo`.
    - In the Host Name field, enter `localhost` (default value).
    - In the Port field, enter `9092` (default value).
    - In the Database User Name field, enter `medrec`.
    - In the Password and Confirm Password fields, enter `medrec`.
20. Click Next.
21. In the Test Database Connection page, ensure that the information to test the connection to the PointBase database is correct.

    The values of the text fields are based on information you have already provided as well as default Pointbase values (such as the Pointbase XA driver class name of `com.pointbase.xa.xaDataSource`).

    Click Test Configuration to test the connection; the message `Connection test succeeded` appears in the Messages pane if you have configured the data source correctly.

    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.
22. Click Next.
23. Select the MedRecServer target.
24. Click Finish.

    The new data source is listed in the Data Sources table.
25. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 3: Create the file stores used by Web Services reliable messaging.

The Web Services reliable messaging feature uses a persistent store to store its messages. The MedRec tutorials describe how to use a file store to store the messages; you can also use a JDBC store although this procedure is not documented in the tutorials.

Note: WebLogic Web Services reliable messaging is a service-to-service feature, which means that one client Web Service invokes another Web Service reliably. Typically each Web Service is deployed to two different WebLogic Server instances; each server instance in turn has its own store-and-forward (SAF) agent and persistent store. However, because these tutorials assume a single-server domain for ease of development, both Web Services are deployed to the same server, and thus use the same SAF agent and persistent store.

1. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services.
3. Click Persistent Stores.
4. In the right pane, click New and choose the Create FileStore option.
5. In the Name field, enter `MedRecWseeFileStore`.
6. In the Target drop-down list, select `MedRecServer`.
7. In the Directory field, enter `medrecWseeFileStore`.
8. Click Finish.
9. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 4: Create a JMS JDBC store.

Persistent stores are used to store persistent messages. This JMS JDBC store uses the non-XA data source you created in Step 2: Create the stand-alone JDBC data sources.

1. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services.

3. Click Persistent Stores.
4. In the right pane, click New and choose the Create JDBCStore option.
5. In the Name field, enter `MedRecJMSJDBCStore`.
6. In the Target drop-down list, select `MedRecServer`.
7. In the Data Source drop-down list, select `MedRecGlobalDataSource`.
8. In the Prefix Name field, enter `MedRec`.
9. Click Finish.
10. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 5: Create JMS servers.

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. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services—Messaging.
3. Click JMS Servers.
4. In the right pane, click New.
5. In the Name field, enter `MedRecJMSServer`.
6. In the Persistent Store drop-down list, select `MedRecJMSJDBCStore`.
7. Click Next.
8. In the Target drop-down list, select `MedRecServer`.
9. Click Finish.
10. In the Summary of JMS Servers page, click New.
11. In the Name field, enter `MedRecWseeJMSServer`.
12. In the Persistent Store drop-down list, select `MedRecWseeFileStore`.
13. Click Next.

14. In the Target drop-down list, select `MedRecServer`.

15. Click Finish.

16. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 6: Create the stand-alone JMS module and queue.

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.

You configure JMS queues by first creating a stand-alone JMS module and then configuring it with a JMS queue, as described in the following procedure.

In addition to the queue that is part of the stand-alone JMS module, the `medrecEar` application uses the following JMS queues configured as part of a packaged JMS module,

- `REGISTRATION_MDB_QUEUE`
- `MAIL_MDB_QUEUE`
- `XML_UPLOAD_MDB_QUEUE`

These application-scoped JMS queues are first registered in the `weblogic-application.xml` deployment descriptor of the `medrecEar` application, and then described in XML files.

1. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services—Messaging.
3. Click JMS Modules.
4. In the right pane, click New.
5. On the Create JMS System Module page:
   - In the Name field, enter `MedRec-jms`.
   - In the Descriptor File Name field, enter `MedRec-jms.xml`.
   - Leave the Location in Domain field blank.
6. Click Next.
7. In the Targets box, check `MedRecServer`.

8. Click Next.
9. Check the Would You Like to Add Resources to This JMS System Module? checkbox.
10. Click Finish.
11. At the bottom of the Configuration tab, above or below the Summary of Resources table, click New.
12. Choose the Queue option.
13. Click Next.
14. In the JMS Destination Properties page:
    - In the Name field, enter `weblogic.wsee.reliability.wseeMedRecDestinationQueue`.
    - In the JNDI Name field, enter `weblogic.wsee.DefaultQueue`.
    - Leave the Template drop-down list to `None`.
15. Click Next.
16. Click Advanced Targeting.
17. To the right of the SubDeployments drop-down list, click Create a New Subdeployment.
18. In the SubDeployment Name field, enter `MedRecWseeJMSServer`.
19. Click OK.
20. In the Targets box, check `MedRecWseeJMSServer`.
21. Click Finish.
22. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 7: Create the Store and Forward (SAF) agent.

The Store-and-Forward (SAF) service enables WebLogic Server to deliver messages reliably between applications that are distributed across WebLogic Server instances. In the MedRec application, the SAF service is used internally by the Web Services that have been configured with reliable messaging.

Note: WebLogic Web Services reliable messaging is a service-to-service feature, which means that one client Web Service invokes another Web Service reliably. Typically each Web

Service is deployed to two different WebLogic Server instances; each server instance in turn has its own SAF agent and persistent store. However, because these tutorials assume a single-server domain for ease of development, both Web Services are deployed to the same server, and thus use the same SAF agent.

1. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services—Messaging.
3. Click Store-and-Forward Agents.
4. In the right pane, click New.
5. On the Store-and-Forward Agent Properties page:
   - In the Name field, enter `MedRecSAFAgent`.
   - In the Persistent Store drop-down list, select `MedRecWseeFileStore`.
   - In the Agent Type drop-down list, select `Both`.
6. Click Next.
7. In the Servers box, check `MedRecServer`.
8. Click Finish.
9. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 8: Add email capabilities to the MedRec application.

WebLogic Server includes the JavaMail API version 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 pre-configure.

1. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, expand MedRecDomain—Services.
3. Click Mail Sessions.

4. In the right pane, click New.
5. In the Name field, enter `mail/MedRecMailSession`.
6. Click OK.
7. In the table of Mail Sessions, click on `mail/MedRecMailSession`.
8. Select the Configuration tab and:
   - In the JNDI Name field, enter `mail/MedRecMailSession`.
   - In the JavaMail Properties text box, enter values for the `mail.user` and `mail.host` properties.

     For example, if you want 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
     ```
9. Click Save.
10. Select the Targets tab.
11. In the Servers box, check `MedRecServer`.
12. Click Save.
13. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 9: Configure the MedRec Custom DBMS Authenticator.

The MedRec Custom DBMS 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. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure window, click MedRedDomain→Security Realms.
3. In the right pane, click the `myrealm` entry of the Realms table.
4. Select the Providers→Authentication tab.
5. Click New.
6. On the Create a New Authentication Provider page:

- In the Name field, enter `MedRecSampleAuthenticator`.
- In the Type drop-down list, select `CustomDBMSAuthenticator`.

7. Click OK.
8. In the Authentication Providers table, click `MedRecSampleAuthenticator`.
9. Select the Configuration→Common tab.
10. In the Control Flag drop-down list, select SUFFICIENT.

    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.

11. Click Save.
12. Select the Configuration→Provider Specific tab.
13. In the Data Source Name field, enter `MedRecGlobalDataSourceXA`.
14. In the Plug-in Class Name field, enter `com.bea.medrec.security.MedRecDBMSPlugin`.

    This plug-in class is physically located in the `MedRecDBMSPlugin.jar` file that you added to WebLogic Server's `CLASSPATH` in Tutorial 1: Creating a WebLogic Domain and Server Instance for Development.

15. Click Save.
16. In the left Domain Structure window, click MedRedDomain→Security Realms.
17. In the right pane, click the `myrealm` entry of the Realms table.
18. Select the Providers→Authentication tab.
19. In the Authentication Providers table, click `DefaultAuthenticator`.
20. In the Control Flag drop-down list, select SUFFICIENT.
21. Click Save.
22. In the left Domain Structure window, click MedRedDomain→Security Realms.
23. In the right pane, click the `myrealm` entry of the Realms table.
24. Select the Providers→Authentication tab.

25. Click Reorder.

    Because WebLogic Server cycles through available Authentication providers, this step reorders 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).

26. Use the arrows in the Authentication Providers list to define the following order of authentication providers:

    a. `DefaultAuthenticator`

    b. `MedRecSamplesAuthenticator`

    c. `DefaultIdentityAsserter`

27. Click OK.

28. In the Change Center, click Activate Changes to update the MedRec server configuration.

Best Practices

- You typically should use an XA JDBC driver to create a JDBC data source.
- JMS JDBC stores do not support XA resource drivers because WebLogic JMS implements its own XA resource. Therefore, do not associate a JDBC data source 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 that uses 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 that 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` application-scoped 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 data source 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 3-4.

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.

The `medrecEar` application has two flavors of Web Services that the `physicianEar` application invokes: reliable and non-reliable. The reliable Web Service uses the store-and-forward agent and filestore internally to ensure that the messages are delivered reliably.

The WebLogic Server security framework cycles through the three configured providers (`DefaultAuthenticator`, `MedRecSamplesAuthenticator`, and `DefaultIdentityAsserter`) during patient authentication.

Related Reading

- *Configuring WebLogic JDBC Resources*
- Configure database connectivity in the *Administration Console Online Help*
- *Configuring and Managing WebLogic JMS*
- Configure Messaging in the *Administration Console Online Help*
- *Configuring and Managing WebLogic Store and Forward*
- *Programming JavaMail with WebLogic Server*
- *Configuring Authentication Providers*

CHAPTER 5

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

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

To put the 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. Open the Administration Console by navigating in a browser to:

   `http://host:7101/console`

   where *host* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.
3. In the left pane, expand MedRecDomain→Environments.
4. Click Servers.
5. In the right pane, in the Servers table, click `MedRecServer(admin)`.
6. Select the Control→Start/Stop tab.
7. At the bottom of the page, in the Server Status table, check `MedRecServer`.
8. Click Shutdown→When Work Completes.
9. Click Yes.

Note: Because you are shutting down the Administration Server, which is what you connect to when you invoke the Administration Console, you will get a browser error once the server actually shuts down.

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 stand-alone 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:

   ```
   prompt> cd c:\bea\user_projects\domains\MedRecDomain\bin
   ```

2. Open the `setDomainEnv.cmd` (Windows) or `setDomainEnv.sh` (UNIX) script in a text editor:

   ```
   prompt> notepad setDomainEnv.cmd
   ```

3. Look for the line that sets 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.

Reboot the server to ensure that it starts up in development mode:

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

   ```
   prompt> 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:

   ```
   <Dec 9, 2005 11:35:57 AM PST> <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 `\autodeploy` 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 you work 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

- Specify the Server Start Mode and JDK in *Creating WebLogic Domains Using the Configuration Wizard*
- Starting and Stopping Servers: Quick Reference in *Managing Server Startup and Shutdown*
- Start and stop servers in the *Administration Console Online Help*

CHAPTER 6

Developing 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.

The preferred BEA method for building applications with WebLogic Server is Apache Ant. Ant is a Java-based build tool. One of the benefits of Ant is that is it is extended with Java classes, rather than with shell-based commands. BEA provides numerous Ant extension classes to help you compile, build, deploy, and package applications in the WebLogic Server split development directory environment.

This tutorial includes the following sections:

Prerequisites

Before starting this tutorial:

- Create the MedRec domain and MedRec server, and start the MedRec administration 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.
- Use the Administration Console to configure the resources needed to run the MedRec applications. See Tutorial 3: Configuring WebLogic Server Resources with the Administration Console.

Procedure

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 install.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`:

```
prompt> mkdir c:\medrec_tutorial
```

Step 2: Unpack the project subdirectories.

BEA provides a `.zip` file that contains the entire source code of the MedRec application; the source code files are used in these tutorials. This `.zip` file also contains a build directory that contains some pre-compiled classes of the MedRec application, as well as the XML files that correspond to procedures in these tutorials.

To populate your project directory with the necessary files and directories:

1. Download the medrec_tutorial.zip (MD5 checksum) file. Save the downloaded file to the `c:\medrec_tutorial` directory.
2. Open up a command window and set your command shell environment with the MedRecDomain environment script:

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

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

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

Step 3: Verify the project directory contents.

Verify that the following files and subdirectories were created:

```
prompt> dir

 Directory of C:\medrec_tutorial

12/09/2005  11:39 AM    <DIR>          .
12/09/2005  11:39 AM    <DIR>          ..
12/09/2005  11:39 AM    <DIR>          build
12/09/2005  11:39 AM    <DIR>          dist
12/09/2005  11:39 AM               493 install.properties
12/09/2005  11:39 AM    <DIR>          lib
12/09/2005  11:39 AM         6,799,498 medrec_tutorial.zip
12/09/2005  11:39 AM    <DIR>          META-INF
12/09/2005  11:39 AM    <DIR>          src
12/09/2005  11:39 AM               996 substitute.xml
```

The `\build` directory contains the 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 (`\clients`) or for a MedRec application (`\medrecEar`, `\physicianEar`, `\initEar`, and `\startBrowserEar` applications).

If you look at the contents of the `\build` directory, you'll notice that many of the classes that make up the MedRec application have already been built for you. In particular, all `medrecEar` and `startBrowserEar` applications, and the Web Services in the `physicianEar` application,

have been pre-compiled. The parts of the `physicianEar` application that have not been pre-compiled are those that you will be working with in these tutorials, namely EJBs and Web applications. This pre-compilation is for your convenience, so you can browse many of the directories and compiled classes of the MedRec application right away without having to build it yourself. Later tutorials demonstrate how to re-compile parts, and then all, of the MedRec application using the WebLogic Ant tasks, such as `wlcompile` and `jwsc`, as part of the normal development process.

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 the appropriate `\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 a few JAR files corresponding to some of the pre-compiled classes in the `\build` directory. The `\dist` directory will eventually store complete, exploded `.ear` directories for the different MedRec applications. The `\dist` directory 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.

The `\lib` directory contains precompiled, third-party `.jar` files that several of the MedRec applications require. This includes supporting `.jar`s 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:

```
prompt> dir src

 Directory of C:\medrec_tutorial\src
```

```
12/09/2005  11:39 AM    <DIR>          .
12/09/2005  11:39 AM    <DIR>          ..
12/09/2005  11:39 AM            19,736 build.html
12/09/2005  11:39 AM             2,938 build.xml
12/09/2005  11:39 AM    <DIR>          clients
12/09/2005  11:39 AM    <DIR>          common
12/09/2005  11:39 AM    <DIR>          initEar
12/09/2005  11:39 AM             4,550 medrec.properties
12/09/2005  11:39 AM    <DIR>          medrecEar
12/09/2005  11:39 AM    <DIR>          physicianEar
12/09/2005  11:39 AM    <DIR>          security
12/09/2005  11:39 AM    <DIR>          startBrowserEar
```

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 `\src` directory 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.

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 `\initEar` subdirectory contains an application that implements and registers a custom MBean that polls the database every 6 seconds to check for new users to be added to the system. You can use this application to register your own custom MBean.
- The `\medrecEar` and `\physicianEar` subdirectories store the main Enterprise Applications that make up the MedRec application suite. These subdirectories use the WebLogic Server 9.1 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 \startupBrowserEar` subdirectory contains the listener 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 install.properties file and run substitute.xml.

The top-level project directory into which you unzipped the ZIP file contains a file called `install.properties` that specifies information local to your environment, such as the name of the computer which hosts WebLogic Server and the MedRec domain directory. After editing this file with information specific to your environment, run the `substitute.xml` Ant file which uses this properties file to substitute your local values for the place-holders in the files you unzipped.

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

   ```
   prompt> notepad c:\medrec_tutorial\install.properties
   ```

2. Edit the `SERVER_HOSTNAME` property to point to the computer that hosts WebLogic Server:

   ```
   SERVER_HOSTNAME=myhost.mycompany.com
   ```

3. If you did *not* use `7101` and `7102` as the administration server port numbers when creating the `MedRecDomain` in Tutorial 1: Creating a WebLogic Domain and Server Instance for Development, edit the `SERVER_PORT` and `SERVER_SSL_PORT` properties to point the appropriate ports:

   ```
   SERVER_PORT=7101
   SERVER_SSL_PORT=7102
   ```

4. Edit the `WL_HOME` property to point to your WebLogic Server installation directory (`c:/bea/weblogic91` by default):

   ```
   WL_HOME=c:/bea/weblogic91
   ```

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

   ```
   MEDREC_HOME_DIR=c:/medrec_tutorial
   ```

6. Edit the `MEDREC_DOMAIN_DIR` property to point to the MedRecDomain directory you created (`c:/bea/user_projects/domains/MedRecDomain` by default):

   ```
   MEDREC_DOMAIN_DIR=c:/bea/user_projects/domains/MedRecDomain
   ```

7. Save and close the `install.properties` file.

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

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

   The substitution script may take a few seconds to complete, at the end of which you should see output similar to the following:

   ```
   Buildfile: substitute.xml
   replace:
   BUILD SUCCESSFUL
   Total time: 23 seconds
   ```

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 by 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

- Creating a Split Development Directory Environment in *Developing Applications with WebLogic Server*
- Overview of WebLogic Server Application Development in *Developing Applications with WebLogic Server*
- Apache Ant in *Developing Applications with WebLogic Server*

CHAPTER 7

Developing the MedRec Applications

Tutorial 6: Understanding the WebLogic Server Split Directory Structure

Several subdirectories in the `medrec_tutorial` project directory—`medrecEar`, `physicianEar`, `startBrowserEar`—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 guides 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 4: Examine the Web Service 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. Note the contents of the `physicianEar` subdirectory:

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

prompt> dir

 Directory of C:\medrec_tutorial\src\physicianEar

12/09/2005  11:39 AM    <DIR>          .
12/09/2005  11:39 AM    <DIR>          ..
12/09/2005  11:39 AM            59,584 build.html
12/09/2005  11:39 AM            10,382 build.xml
```

```
12/09/2005  11:39 AM               592 ejbgen_tutorial.xml
12/09/2005  11:39 AM    <DIR>          META-INF
12/09/2005  11:39 AM    <DIR>          physicianWebApp
12/09/2005  11:39 AM    <DIR>          physSessionEjbs
12/09/2005  11:39 AM    <DIR>          webServices
12/09/2005  11:39 AM               284 wlcompile_tutorial.xml
12/09/2005  11:39 AM               396 wldeploy_tutorial.xml
12/09/2005  11:39 AM             1,015 ws_ejb_client_tutorial.xml
```

As you can see from the directory listing, the Physician application contains a Web Application component (stored in the `physicianWebApp` directory), EJB components (stored in the `physSessionEjbs` directory), and a Web Service (stored in the `webServices` directory). 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 however you want, because the `wlcompile` Ant task automatically determines the type of component during compilation.

The `META-INF` subdirectory holds deployment descriptors for the Enterprise Application itself (`application.xml` and optional `weblogic-application.xml` files).

Note: The various `*_tutorial.xml` files are tutorial-related files.

Step 2: Examine the Web Application component directory structure.

MedRec's Web Applications are developed using Struts, which is an open source Web framework developed by the Apache Jakarta Project.

The source directory structure allows you to easily manage the different file types that constitute a Web Application, such as JSPs and servlets and the files required by the Struts framework. Move to the `physicianWebApp` subdirectory of the `physicianEar` source directory and examine its contents:

```
prompt> cd c:\medrec_tutorial\src\physicianEar\physicianWebApp

prompt> dir

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

12/09/2005  11:39 AM    <DIR>          .
12/09/2005  11:39 AM    <DIR>          ..
12/09/2005  11:39 AM            12,950 Confirmation.html
```

```
12/09/2005  11:39 AM             1,639 Confirmation.jsp
12/09/2005  11:39 AM            41,763 CreatePrescription.html
12/09/2005  11:39 AM             5,389 CreatePrescription.jsp
12/09/2005  11:39 AM            87,839 CreateVisit.html
12/09/2005  11:39 AM            11,961 CreateVisit.jsp
12/09/2005  11:39 AM            14,708 Error.html
12/09/2005  11:39 AM             1,905 Error.jsp
12/09/2005  11:39 AM            32,486 Login.html
12/09/2005  11:39 AM             4,074 Login.jsp
12/09/2005  11:39 AM            15,377 PatientHeader.html
12/09/2005  11:39 AM             2,309 PatientHeader.jsp
12/09/2005  11:39 AM             7,365 PhysicianHeader.html
12/09/2005  11:39 AM               964 PhysicianHeader.jsp
12/09/2005  11:39 AM            30,765 Search.html
12/09/2005  11:39 AM             3,896 Search.jsp
12/09/2005  11:39 AM            29,938 SearchResults.html
12/09/2005  11:39 AM             4,243 SearchResults.jsp
12/09/2005  11:39 AM             5,971 stylesheet.css
12/09/2005  11:39 AM            24,957 ViewProfile.html
12/09/2005  11:39 AM             3,742 ViewProfile.jsp
12/09/2005  11:39 AM            46,175 ViewRecord.html
12/09/2005  11:39 AM             6,707 ViewRecord.jsp
12/09/2005  11:39 AM            37,675 ViewRecords.html
12/09/2005  11:39 AM             5,504 ViewRecords.jsp
12/09/2005  11:39 AM    <DIR>          WEB-INF
```

The top level of the Web Application subdirectory contains the JSPs that make up the application. For each JSP, there is also a corresponding HTML file that simply shows the JSP code in HTML format, with line numbers and color coding for easy reading. These HTML files are included for tutorial purposes only; typically you do not include these types of HTML files in your application. You could also store additional `.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 `\html_files` or `\images`.

Java source files for Web Application components, such as Servlets (also called *actions* in Struts parlance) 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\actions\PhysicianConstants.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`) and the Struts-related files, such as `struts-config.xml`.

The images used in the `physician` Web application are not stored in the `physicianWebApp` directory, but rather in a common directory shared by all the MedRec applications. The `weblogic.xml` deployment descriptor file of the `physician` Web application defines a virtual directory that specifies the exact location of these images, as shown in the following excerpt:

```
<virtual-directory-mapping>
  <local-path>c:/medrec_tutorial/src/common/web</local-path>
  <url-pattern>images/*</url-pattern>
</virtual-directory-mapping>
```

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` metadata annotations to specify the shape and behavior of the EJB, rather than defining them in deployment descriptor files. The `ejbgen` annotations are based on the new JDK 5.0 metadata annotations feature. The `wlcompile` Ant task, when it encounters an `*.ejb` file, invokes the EJBGen utility, which in turn uses these annotations to generate the EJB deployment descriptors automatically when you compile the application.

Step 4: Examine the Web Service directory structure.

Java source files for J2EE Web Service components are stored in subdirectories that reflect the Web Service's package structure. For example, the source for the Physician Web Service (that in

turn invokes the a Web Service of the `medrecEar` application reliably) is stored in `C:\medrec_tutorial\src\physicianEar\webServices\com\bea\medrec\webservices\PhysicianWebServices.java`.

All Web Services in the MedRec application suite are implemented using Java Web Service (JWS) files. A JWS file is the core of your Web Service; it contains the Java code that determines how it behaves. A JWS file is an ordinary Java class file that uses JDK 5.0 metadata annotations to specify the shape and characteristics of the Web Service. The JWS annotations you can use in a JWS file include the standard ones defined by the *Web Services Metadata for the Java Platform* specification (JSR-181) as well as a set of WebLogic-specific ones.

The MedRec application then uses the `jwsc` Web Services Ant task to compile JWS files into a deployable Web Service. As defined by the Enterprise Web Services 1.1 specification (JSR-921), Web Services can be implemented by either plain Java classes (packaged in a Web Application WAR) or a stateless session EJB (packaged in an EJB JAR). The `jwsc` Ant task automatically determines what type of backend component to generate, and then generates all supporting files, such as deployment descriptors, user-defined data type components, and the WSDL file. This means that, similar to EJBs that use `ejbgen` annotations, only the JWS file that describes the Web Service is stored in the `\src` directory.

Note: In this release of WebLogic Server, the `wlcompile` Ant task does not compile Web Services that have been implemented with JWS files. Rather, you must explicitly call the `jwsc` Web Service Ant task to generate the Web Service *before* calling the `wlcompile` task to compile all other components, such as EJBs. Additional details about `jwsc` are provided in Tutorial 11: Creating a J2EE Web Service by Programming a JWS File.

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\*\MyEjbGenEjb.ejb
\myProject\myEar\myEjb\com\*\*.java
\myProject\myEar\myWebService\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`.
- Run the `jwsc` Web Services Ant task (to compile JWS files into Web Services) *before* you run the `wlcompile` Ant task to compile all other components, such as EJBs.

The Big Picture

The MedRec application suite uses three split development directories to hold the source for the `medrecEar`, `physicianEar`, and `startBrowserEar` 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` Ant 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

- Creating a Split Development Directory Environment in *Developing Applications with WebLogic Server*

- Building Applications in a Split Development Directory in *Developing Applications with WebLogic Server*
- Overview of WebLogic Server Application Development in *Developing Applications with WebLogic Server*
- Developing Web Applications, Servlets, and JSPs for WebLogic Server
- Programming WebLogic Server Enterprise JavaBeans
- Programming Web Services for WebLogic Server

CHAPTER 8

Developing the MedRec Applications

Tutorial 7: Compiling Split Development Directory Applications with Ant Tasks

This tutorial explains how to compile Enterprise Application source files with 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:

- Compiling Java Web Service (JWS) files into deployable Web Services. The `wlcompile` Ant task does not currently compile Web Services implemented with JWS files.
- Packaging files from the source and build directories into an EAR file or expanded EAR directory
- Deploying applications

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 copy over the MedRec source files and output directories according to 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.

As you may have noticed if you looked at the contents of the `\build` directory right after unpacking the `medrec_tutorial.zip` file, many classes that make up the MedRec application have already been built for you. This includes all the classes in the `\medrecEar` and `\startBrowserEar` applications, as well as the Web Services in the `\physicianEar` application. This tutorial describes how to compile the classes of the `physicianEar` application that have not been pre-compiled (EJBs and Web application-related) using the `wlcompile` Ant task. You can, of course, recompile any of the pre-built classes as well.

Procedure

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.

Note: In this release of WebLogic Server, the `wlcompile` Ant task does not compile Web Services that have been implemented with Java Web Service (JWS) files. Rather, you

must explicitly call the `jwsc` Web Service Ant task to generate the Web Service *before* calling the `wlcompile` task to compile all other components, such as EJBs. Additional details about `jwsc` are provided in Tutorial 11: Creating a J2EE Web Service by Programming a JWS File.

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:

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

   The top-level of `physicianEar` contains subdirectories for the Web Application, EJB, and Web Service 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:

   ```
   prompt> notepad mybuild.xml
   ```

   Note: If you do not want to enter the `build.xml` file manually, copy the file `wlcompile_tutorial.xml` file, located in the `c:\medrec_tutorial\src\physicianEar` directory, 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 `tutorial`:

   ```
   <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">
          <ejbgen source="1.5" />
     </wlcompile>
   </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.

   The `ejbgen` Ant task specifies the version of the JDK that the EJBGen command should use when processing the `*.ejb` files. The EJBs in MedRec use the new JDK 5.0 metadata

annotation feature, thus you should set the `version` attribute to 1.5 (which is the same as the 5.0 version of the JDK).

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">
            <ejbgen source="1.5" />
       </wlcompile>
     </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:

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

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

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

   prompt> 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

   build:

       [javac] Compiling 1 source file to
   C:\medrec_tutorial\build\physicianEar\APP-INF\classes
      [ejbgen] EJBGen 9.0

      [ejbgen]  Creating
   C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
   ntroller\PhysicianSessionHome.java
      [ejbgen]  Creating
   ```

```
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
ntroller\PhysicianSession.java
   [ejbgen]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\\ejb-jar.xml
   [ejbgen]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\\weblogic-ejb-jar
.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
[wlcompile] Note:
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
ntroller\PhysicianSessionEJB.java uses or overrides a deprecated API.
[wlcompile] Note: Recompile with -Xlint:deprecation for details.
    [javac] Compiling 12 source files to
C:\medrec_tutorial\build\physicianEar\physicianWebApp\WEB-INF\classes

BUILD SUCCESSFUL

Total time: 10 seconds
```

3. If you did not receive the above output, or ran into a problem, you can use the Ant build file provided for this tutorial instead:

   ```
   prompt> 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 session EJBs. Verify that the deployment descriptors were created:

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

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

12/09/2005  11:57 AM    <DIR>          .
12/09/2005  11:57 AM    <DIR>          ..
12/09/2005  11:57 AM             2,301 ejb-jar.xml
12/09/2005  11:57 AM               999 weblogic-ejb-jar.xml
```

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

```
prompt> dir
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\contr
oller

 Directory of
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\contr
oller

12/09/2005  11:57 AM    <DIR>          .
12/09/2005  11:57 AM    <DIR>          ..
12/09/2005  12:00 PM               731 PhysicianSession.class
12/09/2005  12:00 PM             3,795 PhysicianSession.java
12/09/2005  12:00 PM             7,755 PhysicianSessionEJB.class
12/09/2005  12:00 PM            10,769 PhysicianSessionEJB.java
12/09/2005  12:00 PM               326 PhysicianSessionHome.class
12/09/2005  12:00 PM             1,776 PhysicianSessionHome.java
```

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

```
prompt> dir
c:\medrec_tutorial\build\physicianEar\physicianWebApp\WEB-INF\classes\com\
bea\medrec

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

12/09/2005  11:57 AM    <DIR>          .
12/09/2005  11:57 AM    <DIR>          ..
12/09/2005  11:57 AM    <DIR>          actions
```

The `actions` directory stores struts action classes.

Notice that the entire build directory for the Enterprise Application (`c:\medrec_tutorial\build\physicianEar`) contains deployment descriptor files only for the EJB and Web Service components. This is because the EJB and Web Service descriptors are generated using `ejbgen` and Java Web Service (JWS) annotations, respectively. (The Web

Services were pre-compiled as part of the `.zip` file). You can recreate the entire contents of the build directory, including the EJB and Web Services deployment descriptors, by rerunning the build script.

The Enterprise Application and Web Application deployment descriptors (`application.xml`, `weblogic-application.xml`, `web.xml`, and `weblogic.xml`) 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 the following sample excerpt from a `build.xml` file:

```
<wlcompile srcdir="${src.dir}" destdir="${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 source directory.

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

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

It is also useful to note that, prior to compilation, the `wlcompile` Ant task adds the contents of both the source and the `build/`*`earName`*`/APP-INF/lib` and `build/`*`earName`*`/APP-INF/classes` to its CLASSPATH.

Related Reading

- Compiling Applications Using wlcompile in *Developing Applications with WebLogic Server*
- Building Applications in a Split Development Directory in *Developing Applications with WebLogic Server*
- Overview of WebLogic Server Application Development in *Developing Applications with WebLogic Server*
- Apache Ant in *Developing Applications with WebLogic Server*
- Developing Web Applications, Servlets, and JSPs for WebLogic Server
- Programming WebLogic Server Enterprise JavaBeans
- Programming Web Services for WebLogic Server

CHAPTER 9

Developing 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 Servlet 2.4 Deployment Descriptor XML Schema.

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 XML Schema at http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd.

MedRec's Web Applications are developed using Struts, which is an open source Web framework developed by the Apache Struts. The Struts framework has its own configuration file, called `struts-config.xml`, located in the same directory as the Web application deployment descriptor (`WEB-INF`). This configuration file is defined by the Struts 1.2 DTD. These tutorials do not describe the Struts framework; for more information, see the Apache Struts site.

The tutorial includes the following sections:

- Prerequisites

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

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 Split Development Directory Applications with Ant Tasks.

Procedure

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

- Step 1: Examine a web.xml file.
- Step 2: Examine a weblogic.xml file.

Step 1: Examine a web.xml file.

In this section, examine how the `web.xml` file from `mainWebApp` Web Application (of the `medrecEar` Enterprise application) 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`:

   ```
   prompt> notepad
   c:\medrec_tutorial\src\medrecEar\mainWebApp\WEB-INF\web.xml
   ```

2. Note the required element in the heading of the file, which sets the version and encoding:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   ```

3. The elements described in the following steps reside within the `web-app` element that they modify.

   ```
   <web-app xmlns:j2ee="http://java.sun.com/xml/ns/j2ee">

   ....

   </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
     </servlet-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. The `taglib` child element of the `jsp-config` element 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* for a description of the other elements in the web.xml file, such as `filter` and `filter-mapping`.

Step 2: Examine a weblogic.xml file.

In this section, examine the contents of the `weblogic.xml` file that configures the `physicianWebApp` of the `physicianEar` application. 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`:

```
prompt> notepad
c:\medrec_tutorial\src\physicianEar\physicianWebApp\WEB-INF\weblogic.xml
```

2. Note the heading that sets the encoding and references the location of the XML Schema file:

```
<?xml version="1.0" encoding="UTF-8"?>

<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90">
```

This URL is the directory that contains the current WebLogic Server 9.1 XML Schema 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 names of the child elements are defined in `weblogic-web-app.xsd`, whose values can be set within the same `session-descriptor` element:

```
<session-descriptor>
  <timeout-secs>600</timeout-secs>
  <invalidation-interval-secs>60</invalidation-interval-secs>
  <persistent-store-type>
     replicated_if_clustered
  </persistent-store-type>
</session-descriptor>
```

The `timeout-secs` child element sets the number of seconds the server waits before timing out a session.

The second child element, `invalidation-interval-secs`, 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 value assigned to the third child element, `persistent-store-type`, 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.

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

```
<virtual-directory-mapping>
     <local-path>c:/medrec_tutorial/src/common/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://host:7101` and the Web application's context root is `physician`. Users access components of the `physician` Web application relative to `http://host: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* for descriptions of all possible elements of the `weblogic.xml` file.

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. DDInit generates descriptors for JARs, WARs, and EARs—see DDInit.

The Big Picture

The MedRec application contains five Web Applications:

- `physicianWebApp` (in the `physicianEar` application)
- `patientWebApp` (in the `medrecEar` application)
- `adminWebApp` (in the `medrecEar` application)
- `mainWebApp` (in the `medrecEar` application)
- `startBrowserWebApp` (in the `startBrowserEar` application)

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, EJBs, and Web Services, 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, generated into the `C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF` directory, are the standard J2EE EJB deployment descriptor file `ejb-jar.xml`, and the WebLogic Server-specific EJB deployment descriptor file, `weblogic-ejb-jar.xml`.

Similarly, the `physicianEar` also contains a Web Service called `PhysicianWebServices`, whose deployment descriptor files are generated into the `C:\medrec_tutorial\build\physicianEar\PhysicianWebServices\META-INF` directory. The Web Service deployment descriptors include the J2EE standard `webservices.xml`, as well as the WebLogic-specific `weblogic-webservices.xml`. Because the `PhysicianWebServices` service is implemented with an EJB, the META-INF directory also contains EJB-related deployment descriptor files.

`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, Web Service, and application deployment descriptor files and the XML Schema files that they reference.

Related Reading

- `web.xml` Deployment Descriptor Elements
- `weblogic.xml` Deployment Descriptor Elements
- Sun Microsystems XML Schema for `web.xml`
- BEA Systems XML Schema for weblogic.xml
- Apache Struts
- Developing Web Applications, Servlets, and JSPs for WebLogic Server

CHAPTER 10

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

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 using Ant to run the default `build.xml` file:

```
prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
prompt> cd c:\medrec_tutorial\src\physicianEar
prompt> 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. In this procedure you learn how easy it is to deploy Enterprise Applications with this directory structure.

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

   ```
   prompt> cd c:\bea\weblogic91\common\eval\pointbase\tools
   prompt> startPointBase.cmd
   ```

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

   a. Open a new command shell window.

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

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

3. Open another command shell window and set your environment:

   ```
   prompt> c:\bea\user_projects\domains\medrecdomain\bin\setDomainEnv.cmd
   ```

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

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

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

```
prompt> 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 `tutorial`:

```
<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:

```
prompt> 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 ******** -deploy
 [wldeploy] weblogic.Deployer invoked with options:  -noexit -name
tutorial_deployment -source C:\medrec_tutorial\build\physicianEar
-adminurl t3://127.0.0.1:7101 -user weblogic -deploy
 [wldeploy] <Dec 9, 2005 12:13:01 PM PST> <Info> <J2EE Deployment SPI>
<BEA-260121> <Initiating deploy operation for application,
tutorial_deployment [archive:C:\medrec_tutorial\build\physicianEar], to
configured targets.>
 [wldeploy] Task 0 initiated: [Deployer:149026]deploy application
tutorial_deployment on MedRecServer.
 [wldeploy] Task 0 completed: [Deployer:149026]deploy application
tutorial_deployment on MedRecServer.
 [wldeploy] Target state: deploy completed on Server MedRecServer
 [wldeploy]

BUILD SUCCESSFUL
Total time: 30 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:

```
prompt> ant -f wldeploy_tutorial.xml
```

If you receive the following error when trying to deploy `physicianEar`, you might have forgotten to release the configuration when previously using the Administration Console:

```
BUILD FAILED

C:\medrec_tutorial\src\medrecEar\build.xml:256:
weblogic.management.ManagementException: [Deployer:149163]The domain
edit lock is owned by another session in non-exclusive mode - this
deployment operation requires exclusive access to the edit lock and hence
cannot proceed.
```

In this case, be sure you click either Activate Changes or Release Configuration in the Change Center of the Administration Console and then rerun the deploy task.

11. To verify that the application deployed, open a new browser window and enter the URL `http://`*`host`*`:7101/physician`, where *`host`* refers to the computer that hosts `MedRecServer`. If your browser is on the same computer as `MedRecServer`, you can use the URL `http://localhost: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 yet available.

12. You use the `wldeploy` task with 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:

    ```
    prompt> 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:

    ```
    <Dec 9, 2005 12:16:07 PM PST> <Info> <J2EE Deployment SPI> <BEA-260121>
    <Initiating undeploy operation for application, tutorial_deployment
    [archive: null], toconfigured targets.>
    Task 3 initiated: [Deployer:149026]remove application
    tutorial_deployment on MedRecServer.
    Task 3 completed: [Deployer:149026]remove application
    tutorial_deployment on MedRecServer.
    Target state: undeploy completed on Server MedRecServer
    ```

13. Use the `weblogic.Deployer` utility to redeploy the `physicianEar` application in preparation for the next tutorial:

    ```
    prompt> java weblogic.Deployer -adminurl t3://127.0.0.1:7101 -user
    weblogic -password weblogic -deploy -name tutorial_deployment -source
    c:\medrec_tutorial\build\physicianEar
    ```

Best Practices

- You can use the `weblogic.Deployer` tool or its associated Ant task, `wldeploy` to target the build directory to a server running in development mode.
- 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, in particular after running `wlcompile` to regenerate and recompile components. You should generally add deploy, redeploy, and undeploy 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. Similarly, to undeploy, replace `action="redeploy"` with `action="undeploy"`.
- Redeployment is unnecessary for changes to non-generated, non-deployment files such as HTML files and JSPs.

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

- wldeploy Ant Task Reference in *Developing Applications with WebLogic Server*
- weblogic.Deployer Command-Line Reference in *Deploying Applications to WebLogic Server*
- Overview of Common Deployment Scenarios
- Overview of Deployment Tools
- Apache Ant in *Developing Applications with WebLogic Server*

CHAPTER 11

Developing the MedRec Applications

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 (`ejb-jar.xml` and `weblogic-ejb-jar.xml`) and EJB source files, such as the Home interface file, from the main programmer-written EJB bean source file that contains your business logic. In particular, this tutorial uses the `PhysicianSession` EJB from the `physicianEar` application.

EJBGen uses JDK 5.0 metadata annotations in the bean file to generate the deployment descriptor files and the supporting EJB Java source files, such as the Home and Remote interfaces. This means that you only need to program one file that contains your business logic, and use EJBGen-specific annotations inside this file to specify the shape and behavior of the EJB. The EJBGen utility then takes care of generating all supporting files.

The `wlcompile` Ant task automatically invokes the EJBGen utility if it encounters a Java file with a `*.ejb` extension, instead of the normal `*.java` extension. This means that you can mix EJBGen-annotated files with standard Java files in an EAR structure and use a single `wlcompile` Ant task to compile all the code.

All EJB files in the MedRec application are already annotated with EJBGen metadata annotations.

The tutorial includes the following sections:

- Prerequisites

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

Prerequisites

Before starting this tutorial, complete "Tutorial 9: Deploying MedRec from the Development Environment" on page 10-1 so that you have already compiled the `physicianEar` application, started the MedRec server, and deployed the application, including its EJBs.

If you completed up to Tutorial 5 but skipped one or more of the subsequent tutorials so that you have not yet deployed the `physicianEar` application, you can catch up by moving to the `c:\medrec_tutorial\src\physicianEar` subdirectory, setting the environment, and using Ant to run the default `build.xml` file to build the entire application and deploy it to WebLogic Server:

```
prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
prompt> cd c:\medrec_tutorial\src\physicianEar
prompt> ant -f build.xml build deploy.physician.ear
```

Procedure

In the following steps you use the Administration Console to view administrative information about a deployed EJB; view the EJB Java source file that contains the EJBGen metadata annotations; view some of the files that EJBGen generates, and finally use EJBGen to regenerate those files.

- Step 1: Use the Administration Console to view information about a deployed EJB.
- Step 2: View the EJB bean class file that contains the EJBGen metadata annotations.
- Step 3: View the already-generated deployment descriptor and Java files.
- Step 4: Use the EJBGen utility to regenerate the deployment descriptor and EJB files.

Step 1: Use the Administration Console to view information about a deployed EJB.

The procedure is not necessary for using EJBGen; it is provided only as additional information about using the Administration Console to view a deployed EJB.

1. Open the Administration Console by navigating in a browser to:

   `http://host:7101/console`

   where *host* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.

3. In the left Domain Structure window, click MedRecDomain→Deployments.

4. In the right pane, in the Deployments table, expand the `tutorial_deployment` enterprise application.

   A list of all the modules, EJBs, and Web Services contained in the application appears below the application name.

   Note: If you followed the directions in Prerequisites to catch up, then the application is listed with the name `PhysicianEar`.

5. Click `PhysicianSessionEJB` under the EJBs heading.

6. Select the tabs to view information about the EJB. For example, the Configuration tab displays session and transaction information for the `PhysicianSessionEJB`.

Step 2: View the EJB bean class file that contains the EJBGen metadata annotations.

1. Open a command window and move to the directory that contains the Java source file for the `PhysicianSessionEJB`:

   ```
   prompt> cd
   C:\medrec_tutorial\src\physicianEar\physSessionEjbs\com\bea\medrec\cont
   roller
   ```

2. Use an IDE or text editor to view the `PhysicianSessionEJB.ejb` file:

   ```
   prompt> notepad PhysicianSessionEJB.ejb
   ```

Because this Java source file has an `.ejb` extension, the `wlcompile` Ant command automatically uses the EJBGen utility to compile it into a deployable EJB. This is a handy convenience when programming EJBs that use EJBGen annotations because you let the `wlcompile` Ant determine how to process the file.

`PhysicianSessionEJB.ejb` file is the main stateless session EJB bean implementation class file that programmers code and contains the business logic that defines how the EJB behaves. The file implements the WebLogic-specific abstract class `weblogic.ejb.GenericSessionBean`, which is a convenience class similar to the standard J2EE `javax.ejb.SessionBean` interface. The file uses EJBGen metadata annotations that further describe the shape and behavior of the EJB. Some typical annotations include:

- `@Session`—Specifies, at the class-level, that the EJB is of type stateless session. Use attributes to specify characteristics of the EJB, as shown in the following example:

```
@Session(maxBeansInFreePool = "1000",
         initialBeansInFreePool = "0",
         transTimeoutSeconds = "0",
         type = Session.SessionType.STATELESS,
        defaultTransaction = Constants.TransactionAttribute.REQUIRED,
         enableCallByReference = Constants.Bool.TRUE,
         ejbName = "PhysicianSessionEJB")
```

- `@JndiName`—Specifies, at the class-level, the remote or local JNDI name of the EJB. For example:

```
@JndiName(remote = "PhysicianSessionEJB.PhysicianSessionHome")
```

- `@RemoteMethod`—Specifies, at the method-level, which methods are exposed in the generated Remote interface.

See EJBGen Reference for additional information about EJBGen and the full list of annotations.

Step 3: View the already-generated deployment descriptor and Java files.

In Tutorial 7: Compiling Split Development Directory Applications with Ant Tasks, you executed the `wlcompile` Ant task, which in turn executed the EJBGen utility to compile the EJBs of the `physicianEar` application. This procedure describes how to view the deployment descriptor files and supporting Java interfaces that were generated by EJBGen, located in the `\build` directory.

1. Open a command window and move to the directory that contains the deployment descriptor files that were generated by the EJBGen utility for `PhysicianSessionEJB`:

```
prompt> cd
c:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF
```

2. Use a text editor to view the two EJB-related deployment descriptor files:
   - `ejb-jar.xml`—The standard J2EE deployment descriptor that specifies the `PhysicianSessionEJB` bean, its interfaces, and its session and transaction types.
   - `weblogic-ejb-jar.xml`—The WebLogic-specific deployment descriptor that specifies the `PhysicianSessionEJB`-
   -
   - ‘1wq
   - pool and time-out deployment settings.

   For details about the elements used in these files, see the ejb-jar_2_1.xsd and weblogic-ejb-jar.xsd XML Schema files.

3. Move to the directory that contains the compiled EJB classes; the `wlcompile` Ant task also copies the Java source of the EJB files to this directory:

   ```
   prompt> cd
   c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
   ntroller
   ```

4. Use an IDE or text editor to view the Java source files that make up the `PhysicianSessionEJB`. These files include the programmer-created bean class that contains the EJBGen annotations (described in Step 2: View the EJB bean class file that contains the EJBGen metadata annotations.), as well as the following Java files generated by the EJBGen utility:
   - `PhysicianSession.java`—The remote interface of the EJB that describes the signature of its public methods, such as `addRecord`, `getRecordSummary`, and so on. This interface extends the J2EE `javax.ejb.EJBObject` interface.
   - `PhysicianSessionHome.java`—The home interface of the EJB that extends the J2EE `javax.ejb.EJBHome` interface.

Step 4: Use the EJBGen utility to regenerate the deployment descriptor and EJB files.

When you program EJBs using annotations within a split development directory structure, you typically only need to execute the `wlcompile` Ant task over the entire source directory to compile the EJB and generate its supporting files into the build directory. As long as the EJB files that use

annotations have the `*.ejb` extension, the `wlcompile` Ant task automatically knows to execute the EJBGen Ant task on the file.

You can, however, also use the EJBGen utility on its own if you want to generate the EJB deployment descriptors and supporting home and remote interfaces individually, outside of the `wlcompile` Ant task. You can execute the EJBGen utility using either of the following methods:

- As a Java utility whose class name is `com.bea.wls.ejbgen.EJBGen`.
- As an Ant task whose task definition is `com.bea.wls.ejbgen.ant.EJBGenAntTask`.

The following procedure shows how to use the Ant task.

1. Open a command window and set your environment:

   ```
   prompt> c:\bea\user_projects\domains\medrecdomain\bin\setDomainEnv.cmd
   ```

2. Move to the `physicianEar` subdirectory:

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

3. Use a text editor to create a new file, `ejbgen.xml`:

   ```
   prompt> notepad ejbgen.xml
   ```

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

4. Start the `ejbgen.xml` file by defining a project named `ejbgen_tutorial` with a default target `run-ejbgen`:

   ```
   <project name="ejbgen_tutorial" default="run-ejbgen">
   ```

5. Use the `taskdef` task to define the full classname of the EJBGen task:

   ```
     <taskdef name="ejbgen"
              classname="com.bea.wls.ejbgen.ant.EJBGenAntTask" />
   ```

6. Define the main target for executing the EJBGen task on the `PhysicianSessionEJB.ejb` Java file:

   ```
     <target name="run-ejbgen">

         <ejbgen
           source="1.5"
           outputDir =
   "C:\medrec_tutorial\build\physicianEar\physSessionEjbs"
           descriptorDir =
   "C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF"
   ```

```
        forceGeneration = "true"
      >
        <fileset dir="physSessionEjbs/com/bea/medrec/controller"
                 includes="PhysicianSessionEJB.ejb" />
      </ejbgen>

    </target>
```

The `source` attribute specifies that the source EJB bean file uses JDK 1.5 metadata annotations (rather than Javadoc tags used in previous releases of WebLogic Server.) The `outputDir` and `descriptorDir` attributes specify the directories into which EJBGen should generate the home and remote interfaces and the deployment descriptors, respectively. The `forceGeneration` attribute specifies that files should always be generated, even if there have been no changes to the source file. Finally, the Ant-standard `fileset` task specifies the directory which contains the source EJB bean files and the name of the file that EJBGen should process, in this case `PhysicianSessionEJB.ejb`.

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

```
</project>
```

8. Your file contents should now resemble the following:

```
<project name="ejbgen_tutorial" default="run-ejbgen">

  <taskdef name="ejbgen"
           classname="com.bea.wls.ejbgen.ant.EJBGenAntTask" />

  <target name="run-ejbgen">

      <ejbgen
        source="1.5"
        outputDir =
"C:\medrec_tutorial\build\physicianEar\physSessionEjbs"
        descriptorDir =
"C:\medrec_tutorial\build\physicianEar\physSessionEjbs\META-INF"
        forceGeneration = "true"
      >
        <fileset dir="physSessionEjbs/com/bea/medrec/controller"
                 includes="PhysicianSessionEJB.ejb" />
      </ejbgen>

    </target>

</project>
```

   Save the file and exit your text editor.

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

```
prompt> ant -f ejbgen.xml
```

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

```
Buildfile: ejbgen.xml

run-ejbgen:
   [ejbgen] EJBGen 9.0
   [ejbgen]
   [ejbgen]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
ntroller\PhysicianSessionHome.java
   [ejbgen]  Creating
C:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
ntroller\PhysicianSession.java
   [ejbgen]  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

BUILD SUCCESSFUL
Total time: 3 seconds
```

If you did not receive the above output, or ran into a problem, you can use the Ant build file provided for this tutorial instead:

```
prompt> ant -f ejbgen_tutorial.xml
```

10. Verify that the deployment descriptors and home/remote interfaces were indeed regenerated by checking their file creation date and time:

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

    prompt> dir
    c:\medrec_tutorial\build\physicianEar\physSessionEjbs\com\bea\medrec\co
    ntroller\*.java
    ```

    The actual contents of the files is the same as that of the files that were previously generated with the `wlcompile` Ant task.

Best Practices

Use EJBGen to develop the EJB component of your application. You can simplify your EJB development and code maintenance by writing just the bean implementation files and annotating them with EJBGen metadata annotations, 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 EJBGen task is called implicitly by the `wlcompile` task any time `wlcompile` encounters a file with the `*.ejb` extension.

The `PhysicianSessionEJB.ejb` Java file 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\src\physicianEar\physSessionEjbs\com\bea\medrec\controller\PhysicianSessionEJB.ejb` in an IDE or text editor.

For example, the following class-level `@Session` annotation specifies that the class implements a stateless session EJB, and then defines the pool and timeout settings that you see in the generated `weblogic-ejb-jar.xml`:

```
@Session(maxBeansInFreePool = "1000",
         initialBeansInFreePool = "0",
         transTimeoutSeconds = "0",
         type = Session.SessionType.STATELESS,
         defaultTransaction = Constants.TransactionAttribute.REQUIRED,
         enableCallByReference = Constants.Bool.TRUE,
         ejbName = "PhysicianSessionEJB")
public class PhysicianSessionEJB extends GenericSessionBean {

...
```

Related Reading

- EJBGen Reference
- Understanding Enterprise JavaBeans, in *Programming WebLogic Enterprise JavaBeans*
- JDK 5.0 Metadata Annotations
- Configure EJBs in the *Administration Console Online Help*

CHAPTER 12

Developing the MedRec Applications

Tutorial 11: Creating a J2EE Web Service by Programming a JWS File

This tutorial describes how to create a J2EE Web Service, as specified by the Enterprise Web Services 1.1 specification (JSR-921).

The tutorial first looks at the Java Web Service (JWS) file that implements the Web Service, and then shows how to use `jwsc`, the WebLogic Web Service Ant task, to automatically generate all the supporting artifacts that make up the Web Service. The tutorial then shows how to deploy the Web Service and view its WSDL file. Tutorial 12: Invoking a Web Service from a Client Application describes how Web Services can be invoked by a variety of client applications using SOAP.

A *JWS file* is an ordinary Java class file that uses JDK 5.0 metadata annotations to specify the shape and characteristics of the Web Service. The JWS annotations you can use in a JWS file include the standard ones defined by the Web Services Metadata for the Java Platform specification (JSR-181) as well as a set of WebLogic-specific ones.

After you program the JWS file, you use the `jwsc` WebLogic Web Service Ant task to compile the JWS file into either a Java class or a stateless session EJB, as described by the Enterprise Web Services 1.1 specification. You typically do not need to decide this backend implementation of the Web Service; the `jwsc` Ant task picks the optimal implementation based on the JWS annotations you have specified in the JWS file. The `jwsc` Ant task also generates all the supporting artifacts for the Web Service (deployment descriptors, XML Schema representation

of Java user-defined data types, WSDL file), packages everything into an archive file, and creates an Enterprise Application that you can then deploy to WebLogic Server.

The tutorial includes:

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

Prerequisites

Previous tutorials work exclusively with the Physician application, `physicianEar`. This tutorial uses the main MedRec application, `medrecEar`, which contains a variety of Web Services in the `webServices` subdirectory. This tutorial describes how to program the `MedRecWebServices` service, whose JWS file is located in the `C:\medrec_tutorial\src\medrecEar\webServices\com\bea\medrec\webservices` directory. This Web Service provides public operations for the main MedRec services, such as retrieving information about a patient or a particular visit, or updating a patient's profile.

Before starting this tutorial, create the project directory and copy over the source files and output directories using the instructions in Tutorial 5: Creating the MedRec Project Directory.

Procedure

To implement the `MedRecWebServices` service:

- Step 1: View the MedRecWebServices.java JWS file that implements the Web Service.
- Step 2: Create the build file that contains the call to the jwsc Ant task.
- Step 3: Execute the jwsc Ant task to generate the Web Service.
- Step 4: View the generated Web Service artifacts.
- Step 5: Deploy the Web Service and view its WSDL.

Step 1: View the MedRecWebServices.java JWS file that implements the Web Service.

1. Open a command window and move to the directory that contains the MedRecWebServices.java JWS file that implements the MedRecWebServices service:

   ```
   prompt> cd
   C:\medrec_tutorial\src\medrecEar\webServices\com\bea\medrec\webservices
   ```

2. Use an IDE or text editor to view the `MedRecWebServices.java` file:

   ```
   prompt> notepad MedRecWebServices.java
   ```

   The `MedRecWebServices.java` JWS file is a Java file that contains JWS annotations, both standard and WebLogic-specific, that describe the Web Service-specific shape and behavior of the service. The methods of the Java file contain the business logic that implements the public operations of the Web Service; these methods include `getRecord` that returns the record of a patient based on a record ID and `findPatientBySsn` that returns patient information based on the patient's Social Security number.

   The JWS annotations used in `MedRecWebServices.java` include:

   – `@WebService`—Standard class-level annotation that specifies the name of the Web Service, both internal and as it appears in the WSDL file, and the target name space used in the WSDL:

   ```
   @WebService(name = "MedRecWebServicesPortType",
       serviceName = "MedRecWebServices",
       targetNamespace = "http://www.bea.com/medrec")
   ```

   – `@SOAPBinding`—Standard class-level annotations that specifies the type of Web Service, such as document-literal-wrapped (shown in example) or rpc-encoded:

   ```
   @SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
                use=SOAPBinding.Use.LITERAL,
                parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
   ```

   – `@WLHttpTransport`—WebLogic-specific class-level annotation that specifies the name of the port in the WSDL file and the URI used to invoke the Web Service:

   ```
   @WLHttpTransport(portName = "MedRecWebServicesPort",
       contextPath = "ws_medrec",
       serviceUri = "MedRecWebServices")
   ```

   – `@WebMethod`—Standard method-level annotation that specifies which methods of the JWS file will be exposed as public operations of the Web Service.

See Programming the JWS File for detailed information about creating a JWS file and JWS Annotations Reference for the full list of JWS annotations you can use in a JWS file.

Step 2: Create the build file that contains the call to the jwsc Ant task.

After you program the JWS file, you use the `jwsc` Ant task to generate a deployable Web Service.

1. Open a command window and change to the `medrecEar` subdirectory in the MedRec project directory:

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

2. Use a text editor to create a file called `my_webserv.xml` file in the `medrecEar` directory:

   ```
   prompt> 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 treats `c:/medrec_tutorial` as your MedRec project directory

3. Add the following lines to the `my_webserv.xml` file (substituting, if necessary, your actual MedRec project directory for `c:/medrec_tutorial`); the Ant tasks are described at the end of this step:

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

     <path id="jwsc.class.path">

       <pathelement path="${java.class.path}"/>
      <pathelement path="c:/medrec_tutorial/build/medrecEar/sessionEjbs"/>
       <pathelement
   path="c:/medrec_tutorial/build/medrecEar/APP-INF/lib/value.jar" />
       <pathelement
   path="c:/medrec_tutorial/build/medrecEar/APP-INF/lib/utils.jar" />
       <pathelement
   path="c:/medrec_tutorial/build/medrecEar/APP-INF/lib/log4j.jar" />

     </path>

    <taskdef name="jwsc" classname="weblogic.wsee.tools.anttasks.JwscTask"
   />

     <target name="build.ws">

       <jwsc
         srcdir="webServices/com/bea/medrec/webservices"
         sourcepath="webServices"
   ```

```
        destdir="c:/medrec_tutorial/build/medrecEar"

applicationXml="c:/medrec_tutorial/src/medrecEar/META-INF/application.x
ml"
    >
      <classpath refid="jwsc.class.path" />
      <jws file="MedRecWebServices.java" explode="true"/>

    </jwsc>

  </target>

</project>
```

The `<path>` element is used to create a structure, called `jwsc.class.path`, that contains a list of directories and JAR files that will later be added to the CLASSPATH of the `jwsc` Ant task when it compiles the JWS file.

The `<taskdef>` task identifies the full classname of the `jwsc` Ant task.

The `<jwsc>` Ant task uses the following attributes:

- `srcdir`—Full pathname of the top-level directory that contains the MedRecWebServices.java JWS file.
- `sourcepath`—Full pathname of the top-level directory that contains the Java files referenced by the JWS file, such as JavaBeans used as parameters or user-defined exceptions.
- `destdir`—Full pathname of the directory that will contain the compiled JWS files, XML Schemas, WSDL, and generated deployment descriptor files.
- `applicationXml`—Full name and path of the `application.xml` deployment descriptor of the Enterprise Application.

The `<classpath>` child element of `<jwsc>` adds to its CLASSPATH the directories and JAR files previously specified with the `<path>` element. The `<jws>` child element specifies the name of the JWS file to be compiled and that the generated components should be in exploded format, rather than archived in a WAR or JAR file.

Step 3: Execute the jwsc Ant task to generate the Web Service.

After you create the `my_webserv.xml` file, use it to execute the `jwsc` Ant task to generate the Web Service artifacts based on the `MedRecWebServices.java` JWS file:

1. Open a commend window and set your environment using the `MedRecDomain` environment script:

```
prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
```

2. Move to the `medrecEar` directory:

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

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

```
prompt> ant -f my_webserv.xml
```

The `jwsc` Ant task produces output similar to the following to show its progress:

```
Buildfile: my_webserv.xml

build.ws:
     [jwsc] 1 JWS files will be processed.
     [jwsc] Processing JWS:
C:\medrec_tutorial\src\medrecEar\webServices\com\bea\medrec\webservices
\MedRecWebServices.java
     [jwsc] JWS:
C:\medrec_tutorial\src\medrecEar\webServices\com\bea\medrec\webservices
\MedRecWebServices.java Validated.
     [jwsc] Compiling 2 source files to C:\WINDOWS\TEMP\_ptaghb
     [jwsc] Copying 1 file to
C:\medrec_tutorial\build\medrecEar\MedRecWebServices\WEB-INF
     [jwsc] Copying 9 files to
C:\medrec_tutorial\build\medrecEar\MedRecWebServices\WEB-INF
     [jwsc] Copying 4 files to
C:\medrec_tutorial\build\medrecEar\MedRecWebServices\WEB-INF\classes
     [jwsc] Created JWS deployment directory:
C:\medrec_tutorial\build\medrecEar\MedRecWebServices
[AntUtil.deleteDir] Deleting directory C:\WINDOWS\TEMP\_ptaghb

BUILD SUCCESSFUL
Total time: 8 seconds
```

If you did not receive the preceding output, or ran into a problem, you can use the Ant build file provided for this tutorial instead:

```
prompt> ant -f webservices_tutorial.xml
```

Step 4: View the generated Web Service artifacts.

Although you should never update the artifacts generated by the `jwsc` Ant task, it is sometimes useful to view them so as to understand what makes up a deployable Web Service.

Move to the build directory into which the `jwsc` Ant task generated the `MedRecWebServices` artifacts:

```
prompt> cd c:\medrec_tutorial\build\medrecEar\MedRecWebServices
```

The `MedRecWebServicesPortType.java` file is the generated endpoint service interface file; its contents are based on the methods of the JWS file that will be exposed as public operations of the Web Service.

The `WEB-INF` subdirectory contains the generated deployment descriptors, such as `webservices.xml` and `weblogic-webservices.xml`, and the generated WSDL file `MedRecWebServices.wsdl`. Because this Web Service will be packaged into a Web application archive, this directory also contains corresponding descriptors: `web.xml` and `weblogic.xml`. The `WEB-INF\classes` subdirectory contains the compiled Web Service classes, compiled either from the JWS file or from generated Java files (such as the service endpoint interface `MedRecWebServicesPortType.java`).

Step 5: Deploy the Web Service and view its WSDL.

In this section, you deploy the entire MedRec application, which includes the `MedRecWebServices` Web Service, in the same way that you deployed the Physician application in Tutorial 9: Deploying MedRec from the Development Environment. In this tutorial, however, you use the deploy target of the existing `build.xml` file in the `medrecEar` directory rather than creating a new one.

Once the Web Service is deployed, you can view its WSDL file.

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

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

2. Start PointBase, if it is not already running, by executing its start script from another command window:

   ```
   prompt>
   c:\bea\weblogic91\common\eval\pointbase\tools\startPointBase.cmd
   ```

3. Open another command shell and set your environment:

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

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

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

5. Enter the following command to execute the `deploy.medrec.ear` target of the existing `build.xml` file:

```
prompt> ant deploy.medrec.ear
```

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

```
Buildfile: build.xml

deploy.medrec.ear:

 [wldeploy] weblogic.Deployer -noexit -name MedRecEAR -source
C:\medrec_tutorial\build\medrecEar -targets MedRecServer -adminurl
t3://localhost:7101 -user weblogic -password ******** -deploy
-submoduletargets MedRecJMSServer@MedRecAppScopedJMS@MedRecJMSServer
-securityModel CustomRolesAndPolicies
 [wldeploy] weblogic.Deployer invoked with options:  -noexit -name
MedRecEAR -source C:\medrec_tutorial\build\medrecEar -targets
MedRecServer -adminurl t3://localhost:7101 -user weblogic -deploy
-submoduletargets MedRecJMSServer@MedRecAppScopedJMS@MedRecJMSServer
-securityModel CustomRolesAndPolicies
 [wldeploy] <Dec 9, 2005 3:54:33 PM PST> <Info> <J2EE Deployment SPI>
<BEA-260121> <Initiating deploy operation for application, MedRecEAR
[archive: C:\medrec_tutorial\build\medrecEar], to MedRecServer
MedRecJMSServer .>
 [wldeploy] Task 0 initiated: [Deployer:149026]deploy application
MedRecEAR on MedRecJMSServer,MedRecServer.
 [wldeploy] Task 0 completed: [Deployer:149026]deploy application
MedRecEAR on MedRecJMSServer,MedRecServer.
 [wldeploy] Target state: deploy completed on JMS Server MedRecJMSServer
 [wldeploy] Target state: deploy completed on Server MedRecServer
 [wldeploy]

BUILD SUCCESSFUL

Total time: 20 seconds
```

If you do not receive the preceding output, MedRecServer may not have finished starting up, in which case wait until you see the following status in the command window from which you started the server and then rerun the deploy task:

```
<Dec 9, 2005 10:52:20 AM PST> <Notice> <WebLogicServer> <BEA-000360>
<Server started in RUNNING mode>
```

If you receive the following error when trying to deploy `medrecEar`, you might have forgotten to release the configuration when previously using the Administration Console:

```
BUILD FAILED

C:\medrec_tutorial\src\medrecEar\build.xml:256:
weblogic.management.ManagementException: [Deployer:149163]The domain
edit lock is owned by another session in non-exclusive mode - this
```

```
deployment operation requires exclusive access to the edit lock and hence
cannot proceed.
```

In this case be sure you click either Activate Changes or Release Configuration in the Change Center of the Administration Console and then rerun the deploy task.

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

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

   where *host* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL:

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

Best Practices

- Use JWS files to program a WebLogic Web Service. JWS files use JWS annotations, both standard and WebLogic-specific, to describe the shape and behavior of the Web Service. JWS annotations are based on the new JDK 5.0 metadata annotations feature. With this feature, most Web Service information is contained within the JWS file; supporting artifacts such as deployment descriptors and WSDL files are generated by Ant tasks.
- In general, document-literal-wrapped Web Services are the most interoperable type of Web Service. Use the `@SOAPBinding` annotation to specify the type of Web Service. Document-literal-wrapped is the default.
- Use the `jwsc` Ant task to generate all required Web Service artifacts from a JWS file.

The Big Picture

The `com.bea.medrec.webservices.MedRecWebServices` JWS file of the MedRec application contains methods to view and update patient and record information, such as `getRecord()`, `findPatientByLastName()`, `updatePatient()` and so on. The methods of the JWS file that will be exposed as public operations are annotated with the `@WebMethod` annotation. 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` Web Service 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.

The methods of the `MedRecWebServices` JWS file use the following user-defined Java data types as parameters and return values:

- `com.bea.medrec.value.Patient`
- `com.bea.medrec.value.Record`
- `com.bea.medrec.value.RecordsSummary`

Because clients invoke Web Services using SOAP, which uses XML files over the wire to transmit information, these Java data types must also have a corresponding XML Schema data type which the Web Services runtime uses when converting patient and record data between its Java and XML representations. The `jwsc` Ant task, when compiling the JWS file, also automatically generates XML Schemas types of these Java data types. See the `<types>` element in the generated WSDL file to view the XML Schema types.

Once the `MedRecWebServices` Web Service is deployed, 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 using the operations of the Web Service. 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.

Related Reading

- Iterative Development of WebLogic Web Services Starting From Java: Main Steps in *Programming Web Services for WebLogic Server*
- Programming the JWS File in *Programming Web Services for WebLogic Server*
- JWS Annotation Reference in *Programming Web Services for WebLogic Server*
- jwsc Ant Task Reference in *Programming Web Services for WebLogic Server*
- Apache Ant in *Developing Applications with WebLogic Server*
- Web Services Metadata for the Java Platform (JSR-181) Specification
- Enterprise Web Services 1.1 Specification
- Web Services Description Language (WSDL) 1.1 Specification

CHAPTER 13

Developing the MedRec Applications

Tutorial 12: Invoking a Web Service from a Client Application

This tutorial describes how to invoke the `MedRecWebServices` WebLogic Web Service (contained in the `medrecEar` application) that you created in Tutorial 11: Creating a J2EE Web Service by Programming a JWS File 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 client-side artifacts generated by the `clientgen` Ant task 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.

The `clientgen` WebLogic Web Services Ant task generates, from an existing WSDL file, the client artifacts that client applications use to invoke both WebLogic and non-WebLogic Web Services. These artifacts include:

- The Java source code for the Java API for XML-Based RPC (JAX-RPC) Stub and Service interface implementations for the particular Web Service you want to invoke.
- The Java source code for any user-defined XML Schema data types included in the WSDL file.

- The JAX-RPC mapping deployment descriptor file which contains information about the mapping between the Java user-defined data types and their corresponding XML Schema types in the WSDL file.
- A client-side copy of the WSDL file.

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

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. Open a command window and set your environment:

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

2. Move to the root `physicianEar` source directory and execute the `ant` command to compiled its components:

   ```
   prompt> cd c:\medrec_tutorial\src\physicianEar
   prompt> ant
   ```

   Although this tutorial shows how to build a Web Service in the `physicianEar` application, it is assumed that other components of `physicianEar`, such as the value objects, have already been compiled, which is why this step is required.

3. Move to the root `medrecEar` source directory and execute the following `ant` commands to first build `medrecEar` (including its `MedRecWebServices` service that you are going to invoke in this tutorial) and then deploy the application:

```
prompt> cd c:\medrec_tutorial\src\medrecEar
prompt> ant
prompt> ant deploy.medrec.ear
```

Procedure

Each of the following steps shows the code excerpt needed to invoke a Web Service from a different type of client application.

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

Step 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, and then walks you through the EJB code in the `PhysicianSessionEJB` used to invoke the Web Service.

1. Open a command window and set your environment:

```
prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
```

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

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

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

```
prompt> notepad my_webserv_client.xml
```

4. 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">

  <taskdef name="clientgen"
      classname="weblogic.wsee.tools.anttasks.ClientGenTask" />

  <target name="build.ws.client">

    <clientgen
      wsdl="http://localhost:7101/ws_medrec/MedRecWebServices?WSDL"
      destDir="c:/medrec_tutorial/build/physicianEar/APP-INF/classes"
      packageName="com.bea.medrec.webservices.client"

classpath="${java.class.path};c:/medrec_tutorial/build/physicianEar/APP
-INF/lib/value.jar"/>

    <delete includeEmptyDirs="true" failonerror="false" quiet="false">
      <fileset
dir="c:/medrec_tutorial/build/physicianEar/APP-INF/classes/com/bea/medr
ec/value"/>
    </delete>

    <javac
     srcdir="c:/medrec_tutorial/build/physicianEar/APP-INF/classes/com"
      includes="**/*.java"

classpath="${java.class.path};c:/medrec_tutorial/build/physicianEar/APP
-INF/lib/value.jar;c:/medrec_tutorial/build/physicianEar/APP-INF/classe
s"/>

  </target>

</project>
```

The Ant build file first uses the `taskdef` task to specify the full classname of the `clientgen` Ant task.

The Ant build file then calls the `clientgen` Web Services Ant task which generates the client-side artifacts needed to invoke a Web Service. The `wsdl` attribute specifies that the `clientgen` Ant task use the WSDL of the WebLogic Web Service you deployed in Tutorial 11: Creating a J2EE Web Service by Programming a JWS File when generating the artifacts. The `destdir` attribute specifies that `clientgen` generate its artifacts into the `APP-INF/classes` directory of the `physicianEar` build directory. The `APP-INF` directory is part of the WebLogic split development directory environment and is used to ensure that the generated classes are available to other modules, such as the `PhysicianSessionEJB`, in the deployed application. The `packageName` attribute specifies the package into which Java code is generated. Finally, the `classpath` attribute updates the CLASSPATH with the

value.jar JAR file that contains the compiled value-classes of the user-defined data types, such as Patient and Record.

In this release of WebLogic Server, however, there is no way to tell the clientgen Ant task *not* to generate Java representations of the user-defined data types in the WSDL file, or in other words, the same value-classes that already exist in the value.jar file. The classes generated by clientgen do not include additional methods added to the custom value classes that are required by the MedRec application. For this reason, you must execute the delete task to delete the clientgen-generated classes so that they are not inadvertently used by the MedRec application.

Because clientgen generates uncompiled Java classes, the build file then calls the javac task to compile all generated code.

Note: In the preceding Ant build file, it is assumed that the MedRecWebServices WebLogic Web Service is deployed to WebLogic Server and its WSDL is accessible. This is the typical way you always run the clientgen Ant task. However, if you have not yet deployed the Web Service, you can point the wsdl attribute to a static WSDL file, in particular the one that was generated when you compiled the medrecEar application that contains the MedRecWebServices service. To use the static WSDL file, update the wsdl attribute of the clientgen task in my_webserv_client.xml as shown:

```
<clientgen
wsdl="c:/medrec_tutorial/build/medrecEar/MedRecWebServices/WEB-INF/MedRecWebServices.wsdl"
...
```

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

```
prompt> ant -f my_webserv_client.xml
```

The clientgen Ant task shows output similar to the following:

```
Buildfile: my_webserv_client.xml
Trying to override old definition of task clientgen

build.ws.client:

[clientgen] Generating client from
http://localhost:7101/ws_medrec/MedRecWebServices?WSDL ...
[clientgen] Getting partner link
[clientgen] Package name is com.bea.medrec.webservices.client
[clientgen] DestDir is
C:\medrec_tutorial\build\physicianEar\APP-INF\classes
[clientgen] class name is MedRecWebServicesPortType_Stub
[clientgen] service class name is MedRecWebServices
```

```
[clientgen] Porttype name is MedRecWebServicesPortType
[clientgen] service impl name is MedRecWebServices_Impl
   [delete] Deleting 8 files from
C:\medrec_tutorial\build\physicianEar\APP-INF\classes\com\bea\medrec\va
lue
   [delete] Deleted 1 directory from
C:\medrec_tutorial\build\physicianEar\APP-INF\classes\com\bea\medrec\va
lue
    [javac] Compiling 5 source files
    [javac] Note:
C:\medrec_tutorial\build\physicianEar\APP-INF\classes\com\bea\medrec\we
bservices\client\MedRecWebServicesPortType_Stub.java uses unchecked
orunsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL
Total time: 7 seconds
```

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 `medrecEar` application already contains the code needed to invoke the `MedRecWebServices` Web Service.

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

   ```
   prompt> cd
   c:\medrec_tutorial\src\physicianEar\physSessionEjbs\com\bea\medrec\cont
   roller
   ```

   b. Open the `PhysicianSessionEJB.ejb` file in an IDE or text editor:

   ```
   prompt> notepad PhysicianSessionEJB.ejb
   ```

   c. Search for the private method `getMedRecWebServicesPort()`. This method contains the standard Java code that creates a JAX-RPC port of the `MedRecWebServices` Web Service, shown in bold:

   ```
   wsUrl = (String) initCtx.lookup("java:comp/env/webservice/url");
   wsUsername = (String)
   initCtx.lookup("java:comp/env/webservice/username");
   wsPassword = (String)
   initCtx.lookup("java:comp/env/webservice/password");
   logger.debug("MedRec Web Service URL: " + wsUrl);
   MedRecWebServices service = new
   MedRecWebServices_Impl(wsUrl+"?WSDL");
   port = service.getMedRecWebServicesPort(wsUsername, wsPassword);
   ```

   The URL of the WSDL of the deployed `MedRecWebServices`, as well as the user and password of the user that will invoke the Web Service, is defined using the

`@EnvEntries` EJBGen annotation at the beginning of the `PhysicianSessionEJB.ejb` file. The actual URL is:

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

where *host* is the name of the computer hosting MedRec.

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

For example, search for the public method `getRecord`. It contains the following Java code that invokes the `getRecord` operation of the `MedRecWebServices` Web Service by using the `port` instance created in the preceding step:

```
 // Get record from MedRec.
recordVO = port.getRecord(pRecordId.intValue());
```

7. Compile, deploy, and run `PhysicianSessionEJB` as usual.

For information about compiling, see Tutorial 7: Compiling Split Development Directory Applications with Ant Tasks.

Step 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, and then walks you through the actual Java client code you need to write. It is assumed that you know how to write, compile, and run a Java Swing client application.

1. Open a command window and set your environment:

```
prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
```

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

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

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

```
prompt> notepad my_webserv_client.xml
```

4. 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"
>

 <taskdef name="clientgen"
   classname="weblogic.wsee.tools.anttasks.ClientGenTask" />

  <target name="build.ws.client">

    <clientgen
      wsdl="http://localhost:7101/ws_medrec/MedRecWebServices?WSDL"
      destDir="c:/medrec_tutorial/build/swing_client/clientgen"
      packageName="com.bea.medrec.webservices"/>

    <javac
      srcdir="c:/medrec_tutorial/build/swing_client/clientgen"
      destdir="c:/medrec_tutorial/build/swing_client/wsclient"
      includes="**/*.java"/>

    <javac
      srcdir="com"
      destdir="c:/medrec_tutorial/build/swing_client/wsclient"
      includes="**/*.java"

classpath="${java.class.path};c:/medrec_tutorial/build/swing_client/wsc
lient" />

  </target>

</project>
```

The Ant build file first uses the taskdef task to specify the full classname of the clientgen Ant task.

The Ant build file then calls the clientgen Web Services Ant task, which generates the client-side artifacts needed to invoke a Web Service. The wsdl attribute specifies that the clientgen Ant task use the WSDL of the WebLogic Web Service you deployed in Tutorial 11: Creating a J2EE Web Service by Programming a JWS File when generating the artifacts. The destdir attribute specifies that clientgen generate its artifacts into the c:/medrec_tutorial/build/swing_client/clientgen directory. The packageName attribute specifies the package into which Java code is generated.

Because clientgen generates uncompiled Java classes, the build file then calls the javac task to compile all generated code. The second javac task compiles the Swing client itself (whose source code is located in c:\medrec_tutorial\src\clients\com) into the same directory into which the clientgen-generated Java code was compiled.

Note: In the preceding Ant build file, it is assumed that the `MedRecWebServices` WebLogic Web Service is deployed to WebLogic Server and its WSDL is accessible. This is the typical way you run the `clientgen` Ant task. However, if you have not yet deployed the Web Service, you can point the `wsdl` attribute to a static WSDL file, in particular the one that was generated when you compiled the `medrecEar` application that contains the `MedRecWebServices` service. To use the static WSDL file, update the `wsdl` attribute of the `clientgen` task in `my_webserv_client.xml` as shown:

```
<clientgen

wsdl="c:/medrec_tutorial/build/medrecEar/MedRecWebServices/WEB-INF/M
edRecWebServices.wsdl"
...
```

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

   ```
   prompt> ant -f my_webserv_client.xml
   ```

   The `clientgen` Ant task shows the following output:

   ```
   Buildfile: my_webserv_client.xml
   Trying to override old definition of task clientgen

   build.ws.client:
   [clientgen] Generating client from
   http://localhost:7101/ws_medrec/MedRecWebServices?WSDL ...
   [clientgen] Getting partner link
   [clientgen] Package name is com.bea.medrec.webservices
   [clientgen] DestDir is C:\medrec_tutorial\build\swing_client\clientgen
   [clientgen] class name is MedRecWebServicesPortType_Stub
   [clientgen] service class name is MedRecWebServices
   [clientgen] Porttype name is MedRecWebServicesPortType
   [clientgen] service impl name is MedRecWebServices_Impl
       [javac] Compiling 13 source files to
   C:\medrec_tutorial\build\swing_client\wsclient
       [javac] Note:
   C:\medrec_tutorial\build\swing_client\clientgen\com\bea\medrec\webservi
   ces\MedRecWebServicesPortType_Stub.java uses unchecked or unsafe operat

   ions.
       [javac] Note: Recompile with -Xlint:unchecked for details.
       [javac] Compiling 4 source files to
   C:\medrec_tutorial\build\swing_client\wsclient

   BUILD SUCCESSFUL
   Total time: 8 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:

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

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

```
prompt> 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());

MedRecWebServicesPortType port  = ws.getMedRecWebServicesPort();

Patient Patient =

(Patient)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 on the JAX-RPC stub `port`, created in the preceding step:

```
port.updatePatient(patient);
```

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

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

8. Run the application using the `run` target in the existing `build.xml` file:

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

In the application, enter a Social Security number of `123456789` in the Enter Patient SSN field and click Submit. If the MedRec application is deployed and running correctly, you will see information returned about the patient Fred Winner.

Step 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 (Version 1.1) on your computer before you can create and run the C# client. See Microsoft .NET Framework Development Center.

The sample C# client that invokes the `MedRecWebServices` WebLogic Web Service is in the following directory:

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

To run the client, either rebuild it using the .NET IDE, or execute the following already-built application:

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

The already-built client is hard-coded to invoke the `MedRecWebService` deployed at `http://localhost:7001`, so be sure to update the **wsdl location** field of the client with the host and port number of your own MedRecServer.

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 almost all the required client-side artifacts, such as the JAX-RPC stubs.
- After generating the client-side artifacts using `clientgen`, you must compile the generated Java code into class files.
- If you are invoking a Web Service from within a module of an Enterprise application, such as an EJB, BEA recommends you compile the clientgen-generated Java code into the `APP-INF/classes` directory of the application so that all modules of the application can access the classes. It is assumed, of course, that you are using the WebLogic split development directory environment.
- Use the `wsdl` attribute of `clientgen` to generate the client-side artifacts 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: Creating a J2EE Web Service by Programming a JWS File, 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-side artifacts 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 essentially the same way as the EJB as far as using the artifacts generated by `clientgen`.

Related Reading

- Invoking Web Services in *Programming Web Services for WebLogic Server*
- clientgen Ant Task Reference in *Programming Web Services for WebLogic Server*
- Programming WebLogic Enterprise Java Beans
- Java API for XML-Based RPC (JAX-RPC) 1.1 Specification
- Web Services Description Language (WSDL) 1.1 Specification
- Simple Object Access Protocol (SOAP) 1.1 Specification
- Java Swing
- Microsoft .NET Framework Development Center

CHAPTER 14

Developing 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:

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

Prerequisites

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 and stages all of the MedRec applications in the correct order. To run this script:

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

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

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

   ```
   prompt> cd c:\medrec_tutorial\src
   ```

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

   ```
   prompt> 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 following message:

   ```
   build:
        [echo] ####   E N D   M E D R E C   ####

   BUILD SUCCESSFUL
   Total time: 2 minutes 22 seconds
   ```

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 `startBrowserEar` and `initEar` are compiled using the `wlcompile` task.
2. The contents of the `security` directory are compiled using the `javac` task.

3. The contents of `common` are compiled. The `common` directory contains the Java source code 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.
4. 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`.
5. The `physicianEar` application is compiled. The Web Service clients in this application that invoke the Web Services of both the `physicianEar` and `medrecEar` applications are generated from the `.wsdl` files that were previously generated into the `build` directory by the `jwsc` Web Services Ant task.
6. The MedRec application suite client applications, located in the `clients` directory, are compiled.

The `deploy.dev` target of the master `build.xml` file takes an additional step of packaging up the contents of the `build` directory for each application (`initEar`, `medrecEar`, and so on) and creating an exploded directory for each application in the `dist` directory. These exploded directories can be deployed to WebLogic Server in the standard way, outside of the split development directory framework. The next tutorials in this suite (Tutorial 14: Packaging MedRec for Distribution and Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production) go into more detail.

Related Reading

- Compiling Applications Using wlcompile in *Developing Applications with WebLogic Server*
- Building Applications in a Split Development Directory in *Developing Applications with WebLogic Server*

- Overview of WebLogic Server Application Development in *Developing Applications with WebLogic Server*
- Apache Ant in *Developing Applications with WebLogic Server*

Moving to Production

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 `medrecEar` application into a single portable EAR that you can hand off to a production team.

The tutorial includes:

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

Prerequisites

Complete Tutorial 13: Compiling the Entire MedRec Project.

Procedure

- Step 1: Package the medrecEar application as an EAR archive file.
- Step 2: Test the package.

Step 1: Package the medrecEar application as an EAR archive file.

The following procedures create and run a script that packages the contents of the `medrecEar` 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:

   ```
   prompt> c:\bea\user_projects\domains\MedRecDomain\bin\setDomainEnv.cmd
   ```

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

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

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

   ```
   prompt> 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/medrecEar"
                 destdir="c:/medrec_tutorial/build/medrecEar"
                toFile="c:/medrec_tutorial/dist/wlpackage_tutorial.ear" />
   ```

   ```
   </target>
   ```

   See Packaging Applications Using wlpackage 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/medrecEar"
              destdir="c:/medrec_tutorial/build/medrecEar"
             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:

```
prompt> ant -f package.xml
```

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

```
Buildfile: package.xml

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

BUILD SUCCESSFUL
Total time: 1 second
```

   If you do not receive the above output, you may have made a typo in creating the `package.xml` file. If this occurs, try to package using the installed tutorial file:

```
prompt> ant -f wlpackage_tutorial.xml
```

10. To verify that `wlpackage_tutorial.ear` has been created, change to the `MedRec` dist directory and use the `dir` command to view a contents of the directory:

```
prompt> cd c:\medrec_tutorial\dist

prompt> dir wlpackage_tutorial.ear
```

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

```
prompt> jar tf wlpackage_tutorial.ear
```

    The full list of files (over 700) is too long to include in this section, but the list will typically start with the following files:

```
META-INF/
META-INF/MANIFEST.MF
META-INF/application.html
META-INF/application.xml
META-INF/weblogic-application.html
```

```
META-INF/weblogic-application.xml
META-INF/weblogic-diagnostics.html
META-INF/weblogic-diagnostics.xml
adminWebApp/
adminWebApp/ConfirmImport.html
adminWebApp/ConfirmImport.jsp
adminWebApp/CreateAdminSuccessful.html
adminWebApp/CreateAdminSuccessful.jsp
adminWebApp/CreateNewAdmin.html
adminWebApp/CreateNewAdmin.jsp
adminWebApp/Diagnostics.html
adminWebApp/Diagnostics.jsp
adminWebApp/Error.html
adminWebApp/Error.jsp
adminWebApp/Header.html
adminWebApp/Header.jsp
adminWebApp/Home.html
adminWebApp/Home.jsp
adminWebApp/Login.html
adminWebApp/Login.jsp
adminWebApp/Logs.html
adminWebApp/Logs.jsp
adminWebApp/MedRecSchema.html
adminWebApp/MedRecSchema.xsd
adminWebApp/ViewImportRecords.html
adminWebApp/ViewImportRecords.jsp
adminWebApp/ViewPatientRequest.html
adminWebApp/ViewPatientRequest.jsp
adminWebApp/ViewRequests.html
adminWebApp/ViewRequests.jsp
adminWebApp/WEB-INF/
adminWebApp/WEB-INF/classes/
...
```

The `EAR` file you have created contains the `medrecEar` 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, open the Administration Console by navigating in a browser to:

   `http://`*`host`*`:7101/console`

where *host* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.
3. In the middle left-hand pane called Domain Structure, click MedRecDomain→Deployments.

   If you have completed the previous tutorials in sequence, you should see the `physicianEAR` (with the deployment name `tutorial_deployment`) and `MedRecEar` applications already listed in the Deployments table in the right pane. The previous tutorials deployed the two applications within the split development directory environment rather than as a deployable archive. If either of the applications is deployed, uninstall it as follows:

   a. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
   b. In the Deployments table, select the application (such as `MedRecEar`) by checking the box to the left of its application name.
   c. Click Stop→Force Stop Now to ensure that the application is stopped.
   d. Click Yes.
   e. In the Deployments table, select the application again.
   f. Click Delete.
   g. Click Yes.
   h. In the Change Center, click Activate Changes to update the MedRec server configuration.
   i. Repeat the preceding steps, if necessary, to stop and undeploy the `physicianEar` application.
4. Click Lock & Edit, located in the upper left Change Center window of the Administration Console.
5. In the right-hand pane, click Install.
6. Use the Location links to navigate to the `C:\medrec_tutorial\dist` directory.
7. Select `wlpackage_tutorial.ear`.
8. Click Next.

9. In the Choose Targeting Style page, select `Install this deployment as an application`.
10. Click Next until you see the Review Your Choices and Click Finish page.

    Review your choices and click Finish when you are satisfied that your choices are correct.
11. The assistant automatically takes you to the configuration tabs for the deployed application. Select the tabs if you want to view configuration information about the application.
12. In the Change Center, click Activate Changes to update the MedRec server configuration.
13. In the configuration pages of the application, select the Control tab.
14. In the Enterprise Application table, select `wlpackage_tutorial` by checking the box to the left of its name.

    This table also lists the Web applications, EJBs, and Web Services that are packaged in the EAR; expand `wlpackage_tutorial` to see the list.
15. Click Start—Servicing all requests.
16. Click Yes.

    As soon as WebLogic Server has completely started the application, the value in the State column changes from `Prepared` to `Active`. This means that client applications can now start using the `medrecEar` application (which is packaged in the `wlpackage_tutorial` archive).
17. To verify that the application deployed and started, open a new browser window and enter the URL `http://`*`host`*`:7101/patient`, where *`host`* refers to the computer that hosts `MedRecServer`. If your browser is on the same computer as `MedRecServer`, you can use the URL `http://localhost:7101/patient`.

    You should receive the Patient 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 yet available.

Best Practices

Creating an actual `*.ear` archive file of an application with the `wlpackage` Ant task is most useful when you want to distribute or hand off the application to a production team. However, when you actually deploy the application for production, consider deploying 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 15: Using WLST and the Administration Console to Deploy the MedRec Package for Productionfor instructions on deploying MedRec in exploded format.

The Big Picture

In this tutorial, you packaged the `medrecEar` application into a single portable EAR file suitable for handing off to a production team. Because of the `wlpackage` Ant task, the split directory structure for development presents no obstacle to switching to a manageable *single* directory structure for production.

Related Reading

- Deploying and Packaging from a Split Development Directory in *Developing Applications with WebLogic Server*
- Apache Ant in *Developing Applications with WebLogic Server*
- Install an Enterprise application in the *Administration Console Online Help*
- Start a deployed Enterprise application in the *Administration Console Online Help*
- Deploying Applications to WebLogic Server

CHAPTER 16

Moving to Production

Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production

This tutorial describes how to use the WebLogic Scripting Tool (WLST) and 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`, `startBrowserEar`, `physicianEar`, and `initEar` 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.

- `startBrowserEar` contains a single class file that starts the browser when the servlets in the Web Applications are initialized.
- `initEar` contains an application that implements and registers a custom MBean that polls the database every 6 seconds, checking for new users that are added to the system

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

The tutorial includes:

- 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: Configuring WebLogic Server Resources with the Administration Console.
- 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 Split Development Directory Applications with Ant Tasks.
- 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

To deploy the MedRec applications for production:

- Step 1: Start PointBase and the MedRec server.

- Step 2: Delete the applications that have already been deployed.
- Step 3: Deploy medrecEar using WLST.
- Step 4: In the Administration Console, deploy physicianEar, initEar, and startBrowserEar .

Step 1: Start PointBase and the MedRec server.

1. Start PointBase, if it is not already running.

   See Step 1: Start the PointBase database.

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

   From the Windows start menu:

   ```
   Start→Programs→BEA Products→User Projects→MedRecDomain→Start Admin
   Server for WebLogic Server Domain
   ```

   From the command line:

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

Step 2: Delete the applications that have already been deployed.

To ensure that you start from a clean slate, use the Administration Console to delete the various MedRec applications that you might have deployed as part of the preceding tutorials:

1. Open the Administration Console by navigating in a browser to the following URL:

   `http://host:7101/console`

   where *host* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password, and click Log In.

3. In the left Domain Structure pane, click MedRecDomain→Deployments.

4. If the Deployments table in the right pane lists applications from previous tutorials, delete them by following these steps for each application:

   a. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.

b. In the Deployments table, select the deployments from previous tutorials (such as `tutorial_deployment` and `wlpackage_tutorial`) by checking the box to the left of the application name.

c. Click Stop—Force Stop Now to ensure that the application is stopped.

d. Click Yes.

e. In the Deployments table, select the application again.

f. Click Delete.

g. Click Yes.

h. In the Change Center, click Activate Changes to update the MedRec server configuration.

Step 3: Deploy medrecEar using WLST.

The following procedure shows how to use the WebLogic Scripting Tool (WLST) to deploy `medrecEar` to WebLogic Server for production.

The WebLogic Scripting Tool (WLST) is a command-line scripting interface that system administrators and operators use to monitor and manage WebLogic Server instances and domains. The WLST scripting environment is based on the Java scripting interpreter, Jython.

System administrators typically use WLST to:

- Propagate a WebLogic Server domain to multiple destinations.
- Retrieve domain configuration and runtime information.
- Edit the domain configuration and persist the changes in the domain's configuration files.
- Automate domain configuration tasks and application deployment.
- Control and manage the server life cycle.
- Access the Node Manager and start, stop, and suspend server instances remotely or locally, without requiring the presence of a running Administration Server.

WLST functionality also includes the capabilities of the WebLogic Server command-line utilities such as the `wlconfig` Ant task and the `weblogic.Deployer` utility. This procedure first describes how to create a Jython script that includes WLST commands, such as `connect` and `deploy`, and then how to create an Ant build file that invokes the WLST tool and passes it the Jython script file as an argument. You can also use WLST interactively, although this feature is not discussed in this tutorial.

1. Open a command window and set your environment:

   ```
   prompt> c:\bea\user_projects\domains\medrecdomain\bin\setDomainEnv.cmd
   ```

2. Move to the `medrecEar` subdirectory:

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

3. Use a text editor to create a new file called `deploy.py`; this file will contain the WLST commands. For example:

   ```
   prompt> notepad deploy.py
   ```

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

4. Add the following lines to the `deploy.py` file (substituting, if necessary, your actual MedRec project directory for `c:\medrec_tutorial`); the WLST commands are described at the end of this step:

   ```
   userName = "weblogic"
   passWord = "weblogic"
   url="t3://localhost:7101"

   connect(userName, passWord, url)

   progress= deploy(
           'medrecEar',
           'c:\medrec_tutorial\dist\medrecEar',
           targets='MedRecServer',

   subModuleTargets='MedRecJMSServer@MedRecAppScopedJMS@MedRecJMSServer',
           securityModel='CustomRolesAndPolicies' )

   progress.printStatus()

   disconnect()

   exit()
   ```

   The `userName`, `passWord`, and `url` variables simply set up the value of the administration user, password, and the URL used to connect to the Administration Server. The `connect` WLST command connects to the MedRecServer using these variables. The `progress` variable captures a `WLSTProgress` object that defines the status of the deployment. The `deploy` command deploys the `medrecEar` application. In addition to the standard options, such as the source (`c:\medrec_tutorial\dist\medrecEar`) and deployment targets, the deploy command also specifies these additional options: `subModuleTargets` to specify the sub-deployment level of the application-scoped JMS modules included in `medrecEar` and `securityModel` to specify that the `medrecEar` application uses custom security roles and policies. Finally, the `printStatus` method of the `progress` variable prints out the

status of the deployment. The `disconnect` command closes the connection and the `exit` command closes the scripting shell.

5. Save and close the file.
6. Use a text editor to create a new file called `wlst_deploy.xml`. For example:

   ```
   prompt> notepad wlst_deploy.xml
   ```

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

7. Start the `wlst_deploy.xml` file by defining a project named `wlst_tutorial` with a default target `deploy`:

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

8. Define the main target for executing the WLST Java utility; use the `arg` task to specify the deploy.py argument:

   ```
   <target name="deploy">
      <java classname="weblogic.WLST" fork="yes">
        <arg line="deploy.py" />
      </java>
   </target>
   ```

   The Java classname of WLST is `weblogic.WLST`, as specified by the `classname` attribute of the `java` task.

9. Complete the `wlst_deploy.xml` file by closing the project element:

   ```
   </project>
   ```

10. Your file contents should now resemble the following:

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

      <target name="deploy">
        <java classname="weblogic.WLST" fork="yes">
         <arg line="deploy.py" />
        </java>
      </target>

    </project>
    ```

    Save the file and exit your text editor.

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

    ```
    prompt> ant -f wlst_deploy.xml
    ```

You should output similar to the following from WLST:

```
Buildfile: wlst_deploy.xml

deploy:

     [java] Initializing WebLogic Scripting Tool (WLST) ...

     [java] Welcome to WebLogic Server Administration Scripting Shell

     [java] Type help() for help on available commands

     [java] Connecting to weblogic server instance running at
t3://localhost:7101 as username weblogic ...

     [java] Successfully connected to Admin Server 'MedRecServer' that
belongs to domain 'MedRecDomain'.

     [java] Warning: An insecure protocol was used to connect to the
server.
     [java] To ensure on-the-wire security, the SSL port or Admin port
     [java] should be used instead.

    [java] Deploying application from c:\medrec_tutorial\dist\medrecEar
to targets MedRecServer upload=false ...
     [java] <Dec 12, 2005 3:17:27 PM PST> <Info> <J2EE Deployment SPI>
<BEA-260121> <Initiating deploy operation for application, medrecEar
[archive: C:\medrec_tutorial\dist\medrecEar], to MedRecJMSServer
MedRecServer .>
     [java] ...Completed the deployment of Application with status
     [java] Current Status of your Deployment:
     [java] Deployment command type: deploy
     [java] Deployment State       : completed
     [java] Deployment Message     : no message
     [java] Current Status of your Deployment:
     [java] Deployment command type: deploy
     [java] Deployment State       : completed
     [java] Deployment Message     : no message

     [java] Disconnected from weblogic server: MedRecServer

     [java] Exiting WebLogic Scripting Tool ...

    [java] <Dec 12, 2005 3:17:37 PM PST> <Warning> <JNDI> <BEA-050001>
<WLContext.close() was called in a different thread than the one in which
it was created.>

BUILD SUCCESSFUL
Total time: 48 seconds
```

12. Using the Administration Console, verify that `medrecEar` appears in the Deployments table and is in an `Active` state. See the first few steps in Step 2: Delete the applications that have already been deployed. for details.

Step 4: In the Administration Console, deploy physicianEar, initEar, and startBrowserEar .

Deploy the remaining MedRec applications to WebLogic Server as follows. See Step 2: Delete the applications that have already been deployed. for details on invoking the Administration Console in your browser.

1. Click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure pane, click MedRecDomain→Deployments.
3. In the Deployments table in the right pane, click Install.
4. Use the links in the Location field to navigate to `C:\medrec_tutorial\dist`.
5. Select `physicianEar` and click Next.
6. In the Choose Targeting Style window, select `Install this deployment as an application` and click Next.
7. In the Optional Settings window, in the Security section, select `Advanced: Use a custom model that you have configured on the realm's configuration page.`

   This choice ensures that you can use the Administration Console to configure custom security roles and policies for the deployment.
8. Click Next.
9. In the Review Your Choices And Click Finish window, select `No, I will review the configuration later` in the Additional Configuration section.

   Review your other choices to ensure that all are correct. The Summary should look something like the following:

   ```
   Deployment:          C:\medrec_tutorial\dist\physicianEar
   Name:                physicianEar
   Staging mode:        Use the defaults defined by the chosen targets
   Security Model:     Advanced: Use a custom model that you have configured
   on the realm's configuration page.
   ```

10. Click Finish.
11. Repeat steps `2` through `10` to install the `initEar` and `startBrowserEar` applications.

12. In the Change Center, click Activate Changes to update your configuration.
13. In the Deployments table, select the three applications you just installed.
14. Click Start→Servicing all requests.
15. Click Yes.

    As soon as MedRecServer starts the applications, the State column for each application in the Deployments table should say `Active`.

    You might also see the main MedRec informational page appear in your browser.
16. Confirm that the MedRec applications are deployed by navigating to the following login pages in a browser:
    - `http://`*`host`*`:7101/physician`
    - `http://`*`host`*`:7101/patient`
    - `http://`*`host`*`:7101/admin`

    In the preceding URLs, *`host`* refers to the computer that hosts `MedRecServer`. If your browser is on the same computer as `MedRecServer`, you can use `localhost`; for example, `http://localhost:7101/physician`.

    You cannot do much more than look at the login pages right now, because these pages require security that you have not yet configured. This task is described in the next few tutorials.

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 using the WebLogic Scripting Tool (WLST), the command-line tool `weblogic.Deployer`, or Ant scripts that run deployment targets.
- Use WLST to deploy your application using command-line scripts. You can also use this tool to perform a vast number of system administration tasks. The WLST scripting environment is based on the Java scripting interpreter, Jython. In addition to WebLogic scripting functions, you can use common features of interpreted languages, including local variables, conditional variables, and flow control statements. WebLogic Server developers and administrators can extend the WebLogic scripting language to suit their environmental needs by following the Jython language syntax.
- 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 development directory framework allows you to 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, however, 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 directories that contained 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

- Delete an Enterprise application in the *Administration Console Online Help*
- Install an Enterprise application in the *Administration Console Online Help*
- Start a deployed Enterprise application in the *Administration Console Online Help*
- Deploying Applications to WebLogic Server
- WebLogic Scripting Tool
- Apache Ant in *Developing Applications with WebLogic Server*

CHAPTER 17

Moving to Production

Tutorial 16: Creating Users, Groups, and Global Security Roles

This tutorial describes how to create the users, groups, and global security roles that are required by the MedRec application.

After you finish this tutorial, you will be able to log in to all three MedRec Web applications as the appropriate type of user (administrator, patient, or physician) and start using the application.

The tutorial includes:

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

Prerequisites

Before starting this tutorial:

- Create `MedRecDomain` and `MedRecServer`, and start `MedRecServer`. See Tutorial 1: Creating a WebLogic Domain and Server Instance for Development.
- Deploy the enterprise application named `MedRecEar`. See Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production.

Procedure

To create the required users, groups, and security roles using the Administration Console:

- Step 1: Specify security realm settings.
- Step 2: Create groups.
- Step 3: Create users and add the users to groups.
- Step 4: Create global roles and grant the global roles to the groups.

Step 1: Specify security realm settings.

1. With `MedRecServer` running, open the Administration Console by navigating in a browser to:

   `http://`*`host`*`:7101/console`

   where *`host`* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.
3. If you have not already done so, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
4. In the middle left-hand pane called Domain Structure, click MedRecDomain—Security Realms.
5. In the Realms table in the right pane, click `myrealm`.
6. Select the Configuration—General tab.
7. 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*.

8. In the When Deploying Web Applications or EJBs drop-down menu, select `Ignore Roles and Policies From DD`.

This setting indicates that you will set security for Web Application and EJB resources in the Administration Console, not in deployment descriptors. For more information, see Understanding the On Future Redeploys Setting in *Securing WebLogic Resources*.

9. Click Save.
10. In the Change Center, click Activate Changes to update the MedRec server configuration.
11. Restart `MedRecServer`. (See Starting and Stopping Servers: Quick Reference in *Managing Server Startup and Shutdown*.)

Step 2: Create groups.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain—Security Realms.
2. In the Realms table in the right pane, click `myrealm`.
3. Select the Users and Groups—Groups tab.

   The Groups table displays all groups currently defined in the WebLogic Authentication provider's database.
4. Click New.
5. In the Name field, enter `MedRecAdmins`.
6. In the Description field, enter `MedRecAdmins can log on to the MedRec Administrators Web site`.
7. In the Provider drop-down list, select `DefaultAuthenticator` (default value).
8. Click OK.
9. Repeat steps 4 - 8 to create a group named `MedRecPatients`, with a description of `MedRecPatients can log on to the MedRec Patients Web site`, and `DefaultAuthenticator` provider.
10. Repeat steps 5 - 9 to create a group named `MedRecPhysicians`, with a description of `MedRecPhysicians can log on to the MedRec Physician Web site`, and `DefaultAuthenticator` provider.
11. In the Groups table, confirm that the three groups have been added.

Step 3: Create users and add the users to groups.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain—Security Realms.
2. In the Realms table in the right pane, click `myrealm`.
3. Select the Users and Groups—Users tab.

   The Users table displays all users currently defined in the WebLogic Authentication provider's database.
4. Click New.
5. In the Name field, enter `admin@avitek.com`.
6. In the Description field, enter `MedRec administrator`.
7. In the Provider drop-down list, select `DefaultAuthenticator` (default value).
8. In the Password and Confirm Password fields, enter `weblogic`.
9. Click OK to save your changes.
10. In the Users table, click `admin@avitek.com`.
11. Select the Groups tab.
12. In the Available choice box, highlight the `MedRecAdmins` group.
13. Click the highlighted arrow to move the `MedRecAdmins` group from the Available to the Chosen choice box.
14. Click Save.
15. Repeat steps 1 - 14 to create a user named `mary@md.com`, a `MedRec physician` who also uses the `weblogic` password and the `DefaultAuthenticator` provider, and belongs in the `MedRecPhysicians` group.
16. Repeat steps 1 - 14 to create a user named `larry@bball.com`, a `MedRec patient` who also uses the `weblogic` password and the `DefaultAuthenticator` provider, and belongs in the `MedRecPatients` group.
17. Repeat steps 1 - 14 to create a user named `medrec_webservice_user`, a `MedRec Web Service User` who also uses the `weblogic` password and the `DefaultAuthenticator` provider, and belongs in the `MedRecPhysicians` group.

18. Repeat steps 1 - 3 to navigate to the Users table of the `myrealm` security realm to confirm that the three users have been added.

Step 4: Create global roles and grant the global roles to the groups.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain—Security Realms.
2. In the Realms table in the right pane, click `myrealm`.
3. Select the Roles and Policies—Roles tab.

   The Roles table displays all global and scoped roles currently defined in the WebLogic Role Mapping provider's database.
4. In the Roles table, expand Global Roles and click Roles.

   The Global Roles table displays all global roles currently defined in the WebLogic Role Mapping provider's database.
5. Click New.
6. In the Name field, enter `MedRecAdmin`.
7. Click OK.
8. In the Global Roles table, click `MedRecAdmin`.
9. In the Global Role Conditions page, click Add Conditions.
10. In the Choose a Predicate page, select `Group` for the Predicate List.
11. Click Next.
12. In the Group Argument Name field, enter `MedRecAdmins`.
13. Click Add.
14. Click Finish.

    The Role Conditions table includes the following entry:

    ```
    Group MedRecAdmins
    ```
15. Click Save.

16. Repeat steps 1 - 15 to create a global role named `MedRecPatient` and to grant this global role to the `MedRecPatients` group.
17. Repeat steps 1 - 15 to create a global role named `MedRecPhysician` and to grant this global role to the `MedRecPhysicians` group.
18. Repeat steps 1 - 4 to view the Global Roles table and confirm that the three new global roles have been added.

Step 5: Log in to and use the MedRec applications.

Now that you have created all the required users, groups, and roles, you can actually log in to the various MedRec Web applications and start using them. First navigate to the following start page in a browser:

```
http://host:7101/start.jsp
```

In the preceding URL, *`host`* refers to the computer that hosts `MedRecServer`. If your browser is on the same computer as `MedRecServer`, you can use `localhost`; for example: `http://localhost:7101/start.jsp`.

The main MedRec application page appears. Click on the links to log in in to the different Web applications, using the following username/passwords:

- Patients—`larry@bball.com`, `weblogic`
- Administrators—`admin@avitek.com`, `weblogic`
- Physicians—`mary@md.com`, `weblogic`

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 `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.

The Big Picture

The MedRec application has been coded such that only certain roles are allowed to access certain modules, in particular login to Web Applications such as `patient`, `physician`, and `admin`. This tutorial showed you first how to create groups to represent patients, administrators, and physicians, then how to create individual users and assign them to a particular group, and finally, how to map a group to a role. Once this security configuration is in place, you can log in to the applications using the appropriate user.

You might have noticed, however, that in Step 3: Create users and add the users to groups., you did not create an actual patient user. This is because patients, along with their personal information, are stored in the PointBase database and are authenticated using a Custom DBMS Authenticator. The database also stores the group to which the user is assigned. You must, however, use the Administration Console to create the `MedRecPatients` group and the `MedRecPatient` role, and then map the group to the role.

The next tutorials show how to secure specific resources, such as Web applications and EJBs.

Related Reading

- Users, Groups, and Security Roles
- Types of Security Roles: Global Roles and Scoped Roles
- *Securing WebLogic Resources*
- Tutorial 17: Securing URL (Web) Resources Using the Administration Console

- Tutorial 18: Using the Administration Console to Secure Enterprise JavaBean (EJB) Resources

CHAPTER 18

Moving to Production

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

This tutorial describes how to secure URL (Web) resources using the Administration Console. It also provides procedures for creating security policies for URL (Web) resource hierarchies.

After you finish this tutorial, the security for the MedRec application running on the domain you created for these tutorials will be the same as the security configured for the out-of-the-box MedRec application and domain.

The tutorial includes:

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

Prerequisites

Before starting this tutorial:

- Create `MedRecDomain` and `MedRecServer`, and start `MedRecServer`. See Tutorial 1: Creating a WebLogic Domain and Server Instance for Development.

- Deploy the Medrec enterprise application. See Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production.
- Create the required users, groups, and global roles. See Tutorial 16: Creating Users, Groups, and Global Security Roles.

Procedure

To secure URL (Web) resources by using the Administration Console:

Step 1: Invoke the Administration Console in your browser.

1. With `MedRecServer` running, open the Administration Console by navigating in a browser to:

   `http://host:7101/console`

   where *`host`* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.

Step 2: Secure the patient Web Application of medrecEar.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the Modules category, click the `patient` Web application module.

4. Select the Security—Policies tab.
5. Click New.

   An assistant appears that enables you to create a security policy for this particular Web Application or a particular component within the Web Application.
6. In the Create a New Policy URL Pattern page, enter `*.do` in the URL Pattern field.

   The URL pattern of `*.do` will secure all components that have a `.do` suffix.
7. Do not change the default value of the Provider Name field.
8. Click Finish.
9. In the Web Application Module URL Patterns table, click `*.do`.
10. In the Policy Conditions section, click Add Conditions.
11. In the Predicate List drop-down list, select `Role`.
12. Click Next.
13. In the Role Argument Name field, enter `MedRecPatient`.

    This policy specifies that only users with the `MedRecPatient` role are allowed to access these components.
14. Click Add.
15. Click Finish.
16. Click Save.

    The Policy Conditions section includes the entry `Role MedRecPatient`.

    The Overwritten Policy section includes the entry `Group everyone`.
17. Repeat steps 1 - 16 to specify that only the `MedRecPatient` role can access URL resources with the suffix `*.jsp`.

    The Policy Conditions section includes the entry `Role MedRecPatient`.

    The Overwritten Policy section includes the entry `Group everyone`.
18. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `login.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

The Overwritten Policy section includes the entry `Role MedRecPatient`.

The `Anonymous` role, unlike `MedRecPatient`, is a default global role that is predefined in WebLogic Server. This step overrides the security policy you previously defined for all `*.do` URL resources so that every user, regardless of their role, is allowed to view the `login.do` page.

19. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `error.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecPatient`.

20. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `register.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecPatient`.

21. When you are finished, the Web Application Module URL Patterns table for the `patient` Web application should include the following URL Pattern entries:

    - `*.do`
    - `*.jsp`
    - `/error.do`
    - `/login.do`
    - `/register.do`

Step 3: Attempt to access the patient Web application.

1. Open a new Web browser and type the following URL:

   `http://`*`host`*`:7101/patient`

   where *`host`* refers to the computer hosting `MedRecServer`. If your browser is on the same computer, then you can use the URL `http://localhost:7101/patient`.

   The browser prompts you for a username and password.

2. In the username field, type `mary@md.com`, and in the password field, type `weblogic`, then click Login.

The login page returns the error `Invalid User Name and/or Password` and re-prompts you for a username and password. (If this is the first time you use the browser to navigate to this screen, it might also request information about the digital certificate being used by the application.)

3. In the username field, type `larry@bball.com`, and in the password field, type `weblogic`, then click Login.

   The browser displays information for the `larry@bball.com` patient, whose full name is Larry Parrot.

   User `mary@md.com` was denied access because you created a security policy for the `patient` Web Application based on the global security role `MedRecPatient`, which user `larry@bball.com` is granted but user `mary@md.com` is not.

Step 4: Secure the admin Web Application of medrecEar.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the Modules category, click the `admin` Web application module.
4. Select the Security→Policies tab.
5. Click New.

   An assistant enables you to create a security policy for this particular Web Application or a particular component within the Web Application.

6. In the Create a New Policy URL Pattern page, enter `*.do` in the URL Pattern field.

   The URL pattern of `*.do` will secure all components that have a `.do` suffix.

7. Do not change the default value of the Provider Name field.
8. Click Finish.
9. In the Web Application Module URL Patterns table, click `*.do`.
10. In the Policy Conditions section, click Add Conditions.
11. In the Predicate List drop-down list, select `Role`.
12. Click Next.

13. In the Role Argument Name field, enter `MedRecAdmin`.

    This policy specifies that only users with the `MedRecAdmin` role are allowed to access these components.

14. Click Add.

15. Click Finish.

16. Click Save.

    The Policy Conditions section includes the entry `Role MedRecAdmin`.

    The Overwritten Policy section includes the entry `Group everyone`.

17. Repeat steps 1 - 16 to specify that only the `MedRecAdmin` role can access URL resources with the suffix `*.jsp`.

    The Policy Conditions section includes the entry `Role MedRecAdmin`.

    The Overwritten Policy section includes the entry `Group everyone`.

18. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `login.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecAdmin`.

    The `Anonymous` role, unlike `MedRecAdmin`, is a default global role that is predefined in WebLogic Server. This step overrides the security policy you previously defined for all `*.do` URL resources so that every user, regardless of their role, is allowed to view the `login.do` page.

19. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `error.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecAdmin`.

20. When you are finished, the Web Application Module URL Patterns table for the `admin` Web application should include the following URL pattern entries:

    - `*.do`
    - `*.jsp`
    - `/error.do`

— `/login.do`

Step 5: Attempt to access the admin Web application.

1. Open a new Web browser and type the following URL:

   `http://`*`host`*`:7101/admin`

   where *`host`* refers to the computer hosting `MedRecServer`. If your browser is on the same computer, then you can use the URL `http://localhost:7101/admin`.

   The browser prompts you for a username and password.

2. In the username field, type `mary@md.com`, and in the password field, type `weblogic`, then click Login.

   The login page returns the error `Invalid User Name and/or Password` and re-prompts you for a username and password. (If this is the first time you use the browser to navigate to this screen, it might also request information about the digital certificate being used by the application.)

3. In the username field, type `admin@avitek.com`, and in the password field, type `weblogic`, then click Login.

   The browser displays a list of administration tasks.

   User `mary@md.com` was denied access because you created a security policy for the `admin` Web Application based on the global security role `MedRecAdmin`, which user `admin@avitek.com` is granted but user `mary@md.com` is not.

Step 6: Secure the physician Web Application of physicianEar.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `physicianEar`.
3. Under the Modules category, click the `physician` Web application module.
4. Select the Security→Policies tab.
5. Click New.

   An assistant enables you to create a security policy for this particular Web Application or a particular component within the Web Application.

6. In the Create a New Policy URL Pattern page, enter `*.do` in the URL Pattern field.

   The URL pattern of `*.do` will secure all components that have a `.do` suffix.

7. Do not change the default value of the Provider Name field.
8. Click Finish.
9. In the Web Application Module URL Patterns table, click `*.do`.
10. In the Policy Conditions section, click Add Condition.
11. In the Predicate List drop-down list, select `Role`.
12. Click Next.
13. In the Role Argument Name field, enter `MedRecPhysician`.

    This policy specifies that only users with the `MedRecPhysician` role are allowed to access these components.

14. Click Add.
15. Click Finish.
16. Click Save.

    The Policy Conditions section includes the entry `Role MedRecPhysician`.

    The Overwritten Policy section includes the entry `Group everyone`.

17. Repeat steps 1 - 16 to specify that only the `MedRecPhysician` role can access URL resources with the suffix `*.jsp`.

    The Policy Conditions section includes the entry `Role MedRecPhysician`.

    The Overwritten Policy section includes the entry `Group everyone`.

18. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `login.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecPhysician`.

    The `Anonymous` role, unlike `MedRecPhysician`, is a default global role that is predefined in WebLogic Server. This step overrides the security policy you previously defined for all `*.do` URL resources so that every user, regardless of their role, is allowed to view the `login.do` page.

19. Repeat steps 1 - 16 to specify that the `Anonymous` role can access the specific URL resource called `error.do`.

    The Policy Conditions section includes the entry `Role Anonymous`.

    The Overwritten Policy section includes the entry `Role MedRecPhysician`.

20. When you are finished, the Web Application Module URL Patterns table for the `physician` Web application should include the following URL pattern entries:

    - `*.do`
    - `*.jsp`
    - `/error.do`
    - `/login.do`

Step 7: Attempt to access the physician Web application.

1. Open a new Web browser and type the following URL:

   `http://`*`host`*`:7101/physician`

   where *`host`* refers to the computer hosting `MedRecServer`. If your browser is on the same computer, then you can use the URL `http://localhost:7101/physician`.

   The browser prompts you for a username and password.

2. In the username field, type `larry@bball.com`, and in the password field, type `weblogic`, then click Login.

   The browser returns an error.

3. Navigate to the `http://host:7101/physician` page again, and this time enter `mary@md.com` in the username field and `weblogic` in the password field, then click Login.

   The browser displays a search page to look up patient information.

   User `larry@bball.com` was denied access because you created a security policy for the `physician` Web Application based on the global security role `MedRecPhysician`, which user `mary@md.com` is granted but user `larry@bball.com` is not.

Best Practices

- Create policy statements based on your organization's established business procedures.

- When creating new security policies, look for policy statements in the Overwritten Policy 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 resource (`login.do`) in a Web application overrides a security policy on a group of resources (`*.do`). 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).
- Make sure 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.
- Make 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 various URL (Web) resources using the same security as that of the out-of-the-box MedRec application.

Related Reading

- Types of WebLogic Resources
- Techniques for Securing URL (Web) and EJB Resources
- Prerequisites for Securing URL (Web) and EJB Resources
- Tutorial 16: Creating Users, Groups, and Global Security Roles
- Tutorial 18: Using the Administration Console to Secure Enterprise JavaBean (EJB) Resources

Moving to Production

Tutorial 18: Using the Administration Console to Secure Enterprise JavaBean (EJB) Resources

This tutorial describes how to secure Enterprise JavaBean (EJB) resources by 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.

Warning: This tutorial does *not* mimic the security configuration of the out-of-the-box MedRec application. Rather, it walks you through a series of steps that secure different levels of EJBs. After each step, you attempt to access the EJB resource to see the effect of implementing the security policy. It does not necessarily reflect a real-life scenario; the procedures just show you what is possible. At the end of the tutorial consider deleting the security policies you created so that the application returns to its correct security configuration. The configured security policies actually interfere with the correct functioning of the MedRec application.

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.
- Deploy the Medrec enterprise application. See Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production.
- Create the required users, groups, and global roles. See Tutorial 16: Creating Users, Groups, and Global Security Roles.

Procedure

To secure Enterprise JavaBean (EJB) resources by using the Administration Console:

Step 1: Create scoped roles and grant the scoped roles to groups.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the Modules category, click the `sessionEjbs` EJB module.
4. Select the Security→Roles tab.

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.

5. Click New.
6. In the Name field, enter `MedRecSessionEJBPatient`.
7. Do not change the default value of the Provider Name field.
8. Click OK.
9. In the EJB Module Roles table, click `MedRecSessionEJBPatient`.
10. In the Role Conditions section, click Add Conditions.
11. In the Predicate List drop-down list box, select `Group`.
12. Click Next.
13. In the Role Argument Name field, enter `MedRecPatients`.

    Note: You created the `MedRecPatients` group as part of "Tutorial 16: Creating Users, Groups, and Global Security Roles" on page 17-1. Recall that user `larry@bball.com` is the only user in this group.

14. Click Add.
15. Click Finish.
16. Click Save.

    The Role Conditions sections includes the entry `Group MedRecPatients`.

17. Repeat steps 1- 16 to create the scoped role named `MedRecSessionEJBAdmin` and grant this scoped role to the `MedRecAdmins` group.

Step 2: Secure the sessionEjbs JAR.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the Modules category, click the `sessionEjbs` EJB module.

4. Select the Security—Policies tab.

   From this page you can create a security policy at the EJB JAR level, which includes all EJBs within the JAR, and all methods within those EJBs.

5. Click Add Conditions.

6. In the Predicate List drop-down list, select `Role`.

7. Click Next.

8. In the Role Argument Name field, enter `MedRecSessionEJBPatient`.

9. Click Add.

10. Click Finish.

11. Click Save.

    The Policy Conditions section includes the entry `Role MedRecSessionEJBPatient`.

    The Overwritten Policy section includes the entry `Group everyone`.

    By defining this security policy for the `sessionEjbs` JAR, you are overriding any security policies that have already been defined for the EJB resource type. For example, if you had previously specified that the entire `medrecEar` application can be accessed only by the `MedRecAdmin` role, then the Overwritten Policy section would include the text `Role MedRecAdmin`.

    However, in this case you are only overwriting the default security policy (`Group everyone`).

    For more information about default security policies, see Default Security Policies in *Securing WebLogic Resources*.

Step 3: Attempt to access an EJB in the SessionEJB JAR.

1. Open a new Web browser and type `http://`*`host`*`:7101/admin`, where *`host`* refers to the computer hosting `MedRecServer`. If your browser is on the same computer, then you can use the URL `http://localhost:7101/admin`.

2. In the User Name field, type `admin@avitek.com`, and in the Password field, type `weblogic`, then click Login.

   Remember that in Tutorial 17: Securing URL (Web) Resources Using the Administration Console, you granted the `MedRecAdmin` role permission to access the `admin` Web Application. Because `admin@avitek.com` has been assigned the `MedRecAdmin` role (via

the `MedRecAdmins` group), the `admin@avitek.com` user is thus allowed to log in to the `admin` Web Application.

3. On the Administration Tasks page, click the View Pending Requests link.

   The browser returns the following error:

   ```
   [EJB:010160]Security Violation: User: 'admin@avitek.com' has
   insufficient permission to access EJB: type=<ejb>,
   application=medrecEar, module=sessionEjbs, ejb=AdminSessionEJB,
   method=findNewUsers, methodInterface=Remote, signature={}.
   ```

   The error is displayed because access to the `findNewUsers()` method in `AdminSessionEJB`, an EJB within the `sessionEjbs` 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 therefore is not granted access.

Step 4: Secure the AdminSessionEJB.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the EJBs category, click the `AdminSessionEJB` EJB module.
4. Select the Security→Policies tab.

   From this page you can create a security policy for the specified EJB, either at the EJB level (meaning the security policy will apply to all methods within the EJB), or a particular method within the EJB.
5. In the EJB Component Methods drop-down list, ensure that the default `ALL` is selected.
6. Click Add Conditions.
7. In the Predicate List drop-down list, select `Role`.
8. Click Next.
9. In the Role Argument Name field, enter `MedRecSessionEJBAdmin`.
10. Click Add.
11. Click Finish.

12. Click Save.

    The Policy Conditions section includes the entry `Role MedRecSessionEJBAdmin`.

    The Overwritten Policy section includes the entry `Role MedRecSessionEJBPatient`.

    By defining this security policy for `AdminSessionEJB`, you are overriding the security policy that has already been defined for the EJB JAR in Step 2: Secure the sessionEjbs JAR. Specifically, you are overriding the inherited policy statement of `Role MedRecSessionEJBPatient`.

Step 5: Attempt to access AdminSessionEJB.

Repeat steps 1 - 3 in "Step 3: Attempt to access an EJB in the SessionEJB JAR." on page 19-4.

Instead of displaying the error page, the browser now displays the list of pending requests.

This result occurs because user `admin@avitek.com` has been 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 6: Secure the findNewUsers() EJB method.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the EJBs category, click the `AdminSessionEJB` EJB module.
4. Select the Security→Policies tab.

   From this page you can create a security policy for the specified EJB, either at the EJB level (meaning the security policy will apply to all methods within the EJB), or a particular method within the EJB.
5. Using the EJB Component Methods drop-down list, select `findNewUsers() - REMOTE`.
6. Click Add Conditions.
7. In the Predicate List drop-down list, select `Role`.
8. Click Next.
9. In the Role Argument Name field, enter `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.

10. Click Add.
11. Click Finish.
12. Click Save.

    The Policy Conditions section includes the entry `Role MedRecSessionEJBPatient`.

    The Overwritten Policy section includes the entry `Role MedRecSessionEJBAdmin`.

    By defining this security policy on the `findNewUsers()` method, you are overriding the security policy that has already been defined for `AdminSessionEJB` in "Step 4: Secure the AdminSessionEJB.." Specifically, you are overriding the inherited policy statement of `Role MedRecSessionEJBAdmin` that is shown when `ALL` is selected from the EJB Component Methods drop-down list.

Step 7: Attempt to access the findNewUsers() EJB method.

Repeat steps 1 - 3 in "Step 3: Attempt to access an EJB in the SessionEJB JAR." on page 19-4.

The browser displays an error page. 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.

Step 8: Optionally, remove the EJB security policies.

The security policies configured in this tutorial do not mimic the out-of-the-box security configuration of the MedRec application, but rather, simply provide examples of how you can secure EJB policies if you choose to do so in your own applications. The configured security policies actually interfere with the correct functioning of the MedRec application, so if you want to run the application without problems, you should remove the security policies as described in the following procedure.

1. In the left Domain Structure pane of the Administration Console, click MedRecDomain→Deployments.
2. In the Deployments table in the left pane, expand `medrecEar`.
3. Under the Modules category, click the `sessionEjbs` EJB module.

4. Select the Security→Policies tab.
5. In the Policies Conditions table, check `Role MedRecSessionEJBPatient`.
6. Click Remove.
7. Click Save.
8. In the left pane, click MedRecDomain→Deployments.
9. In the Deployments table in the right pane, navigate to the `AdminSessionEJB` EJB module of the `medrecEar` application.
10. Select the Security→Policies tab.
11. In the EJB Component Methods drop-down list, select `ALL`.
12. In the Policy Conditions table, check `Role MedRecSessionAdmin`.
13. Click Remove.
14. Click Save.
15. In the EJB Component Methods drop-down list, select `findNewUsers() - REMOTE`.
16. In the Policy Conditions table, check `Role MedRecSessionEJBPatient`.
17. Click Remove.
18. Click Save.

Best Practices

- Do not use blank spaces, commas, hyphens, or any characters in this comma-separated list for scoped role names: \t, < >, #, |, &, ~, ?, ( ), { }. 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 security role names are singular.
- Try to avoid creating 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 `Role` policy condition.
- Scoped roles can be used in security policies from the level in the hierarchy where they are defined and below.

- Create policy statements based on your organization's established business procedures.
- When creating new security policies, look for policy statements in the Overwritten 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. The examples here do *not* reflect the security configuration of the out-of-the-box 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

- Types of WebLogic Resources
- Techniques for Securing URL (Web) and EJB Resources
- Prerequisites for Securing URL (Web) and EJB Resources
- Tutorial 16: Creating Users, Groups, and Global Security Roles
- Tutorial 17: Securing URL (Web) Resources Using the Administration Console

Moving to Production

Tutorial 19: Creating a Deployment Plan and Redeploying the MedRec Package

This tutorial shows how to use the Administration Console to create a deployment plan to store changes to a deployment property of the `medrecEar` application.

A WebLogic Server *deployment plan* is an optional XML document that resides outside of an application archive and configures an application for deployment to a specific WebLogic Server environment. A deployment plan works by setting deployment property values that would normally be defined in an application's WebLogic Server deployment descriptors, or by overriding property values that are already defined in a WebLogic Server deployment descriptor.

Deployment plans are created and owned by the administrator or deployer for a particular environment, and are stored outside of an application archive or exploded archive directory. As a best practice, BEA recommends storing each deployment plan for a single application in its own plan subdirectory of the application's root directory.

Deployment plans help the administrator easily modify an application's WebLogic Server configuration for deployment into to multiple, differing WebLogic Server environments without modifying the deployment descriptor files included in the application archive. For example, a deployment plan enables you to deploy an application to multiple domains, or to multiple target servers and clusters within the same domain, that have different configurations. To deploy the application to a new environment, an administrator simply creates or uses a new deployment plan as necessary.

The tutorial also shows how to redeploy the MedRec applications to MedRecServer in a production environment. The MedRec applications are contained in the `dist` directory, packaged in four 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:

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

Prerequisites

Before starting this tutorial:

- Work through Tutorial 14: Packaging MedRec for Distribution.
- Work through Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production, and have the package currently deployed to `MedRecServer`.

Procedure

To create and update deployment plans and to redeploy the application:

- Step 1: Update a deployment property and save the new configuration to a deployment plan.
- Step 2: Redeploy the entire application.

Step 1: Update a deployment property and save the new configuration to a deployment plan.

Use the Administration Console to modify certain deployment properties in applications that are deployed as exploded archive files. When you save the new values for these deployment properties, the Administration Console automatically updates the deployment plan associated with the application, or creates a new one if one does not already exist.

In this procedure, you will change the number of seconds the `patient` Web Application of `medrecEar` remains idle before timing out.

1. Open the Administration Console by navigating in a browser to:

   `http://`*`host`*`:7101/console`

   where *`host`* refers to the computer on which MedRecServer is running. If your browser is on the same computer as MedRecServer, then you can use the URL `http://localhost:7101/console`.

2. Specify `weblogic` for both the username and password and click Log In.
3. Click Lock & Edit, located in the upper left Change Center window of the Administration Console.
4. In the left Domain Structure pane, click MedRecDomain→Deployments.
5. In the Deployments table in the right pane, expand `medrecEar`.
6. Under the Modules section, click the `patient` Web App module.
7. Select the Configuration→General tab.
8. In the Session Timeout (in seconds) field, enter `3600`.
9. Click Save.
10. In the Save Deployment Plan page, navigate to, and then select, the directory into which you will store the deployment plan.

    For example, if you want to store the deployment plan in the same root directory as the production-ready `medrecEar` application is stored, navigate to the `c:\medrec_tutorial\dist` directory and select `medrecEar`.

11. Click Finish.

    The Administration Console creates a file called `Plan.xml` in the directory you specified. This XML file contains the deployment plan that is associated with the `medrecEar` application.

12. In the Change Center, click Activate Changes to update the `medrecEar` configuration.
13. To view the new deployment plan:

    a. In the left Domain Structure pane, click MedRecDomain→Deployments.

    b. In the Deployments table in the right pane, click `medrecEar`.

c. Select Deployment Plan→Tuning Parameters.

d. Expand Modules, if it is not already expanded.

e. Click on `patient`.

   The Tunable Deployment Plan Variables table lists the property you just updated: `timeoutSecs`.

Step 2: Redeploy the entire application.

In this procedure it is assumed that MedRec is deployed to a currently running instance of `MedRecServer` and that you have made changes to the application that you want to redeploy. Follow these steps to update a deployed application whose class files or generated deployment descriptor files have changed.

1. In the Administration Console, click Lock & Edit, located in the upper left Change Center window of the Administration Console.
2. In the left Domain Structure pane, click MedRecDomain→Deployments.

   The Deployments table in the right pane displays all deployed applications, which include the `medrecEar`, `physicianEar`, `initEar`, and `startBrowserEar` applications you deployed in Tutorial 15: Using WLST and the Administration Console to Deploy the MedRec Package for Production.
3. In the Deployments table, select the `medrecEar` application by checking the box to the left of its name.
4. Click Update.
5. In the Locate New Deployment Files, click Next without making any changes.

   The Locate New Deployment Files pages allows you to specify a completely new source or deployment plan directory if you want to redeploy a new version of the application. For this tutorial, however, redeploy the application using the current files.
6. Review your choices, then click Finish.

   If the application has changed since the last time it was deployed, the State column for `medrecEar` in the Deployments table might change from `Active` to `Redeploy Initializing`.
7. In the Change Center, click Activate Changes; the state of the application should change to `Active`.

8. Optionally repeat steps 2 through 7 for each of the remaining applications: `physicianEar`, `initEar`, and `startBrowserEar`.

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 use the Administration Console to update deployment properties of an application, the console automatically creates a deployment plan and associates it with the application (or updates an existing one). A deployment plan is an XML document that defines an application's WebLogic Server deployment configuration for a specific WebLogic Server environment. A deployment plan resides outside of an application's archive file, and can apply changes to deployment properties stored in the application's existing WebLogic Server deployment descriptors. Use deployment plans to easily change an application's WebLogic Server configuration for a specific environment without modifying existing deployment descriptors. Multiple deployment plans can be used to reconfigure a single application for deployment to multiple, differing WebLogic Server environments.

If you have only modified static files, it is probably possible to refresh the files without redeploying the entire application; in this case you would use the weblogic.Deployer utility rather than the Administration Console. See Updating Static Files in a Deployed Application in *Deploying Application to WebLogic Server*.

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 Managing Deployed Applications in *Deploying Applications and Modules*.

Related Reading

- Update (redeploy) an Enterprise application in the *Administration Console Online Help*
- Update a Deployment Plan in the *Administration Console Online Help*
- Understanding WebLogic Deployment
- Updating Applications in a Production Environment