APPENDIX A

Apache Derby

Apache Derby is a 100% Java standards–based relational database with a small footprint. Derby is useful for unit testing and small applications. In order to follow the examples contained within this book, you will need such a relational database. The Apache Derby database is an ideal candidate because it is open source so everybody has access to it, it is relatively easy to install, and it requires almost no administration.

This appendix provides a short history and an architectural overview of Derby, followed by instructions on how to install and configure Derby as a client/server database for the purposes of running the examples in this book.

History

Apache Derby started as the commercial database known as Cloudscape. It was one of the first available 100% Java databases. It gained popularity with early adopters of J2EE because an evaluation version was bundled with the J2EE Software Development Kit (SDK). The J2EE SDK was frequently used as a way of learning J2EE, until open source J2EE application servers became available and commercial vendors started offering free developer licenses. The J2EE SDK was a popular learning tool because it included a free reference implementation of an application server and the Cloudscape database. This combination enabled developers to quickly and easily get a development environment set up so they could focus on learning how to build J2EE applications.

Later IBM acquired the Cloudscape database in its acquisition of Informix and discontinued the practice of including Cloudscape with the J2EE SDK. IBM already had a family of databases under the name DB2, which fills the niche of high-transactional enterprise databases. IBM assimilated Cloudscape into the family by making it more compatible with the DB2 series. With the addition of Cloudscape, IBM had an offering for applications that don't need a large and expensive DB2 database. Yet using Cloudscape provides a simple migration path to DB2 as an application grows.

In the fall of 2004, IBM donated the Cloudscape code base to the Apache Software Foundation (ASF). At this time, it was renamed Derby. The ASF accepted the donation and immediately made it available in the Apache Incubator (`http://incubator.apache.org`), the location where all external donations begin. When the Derby project (`http://incubator.apache.org/derby/`) is deemed ready, it will be promoted to its host project, Apache DB (`http://db.apache.org`). IBM plans on continuing to sell Cloudscape and service contracts. However, Cloudscape will now be based on Derby.

Architecture

Derby can run as either an embedded database or a networked server. When run as an embedded database, only a single JVM can access the database at a time. The network server mode follows the traditional client/server model by running as a separate process that enables access from multiple JVMs and users.

Embedded Mode

The majority of enterprise applications have functional requirements requiring accessibility from multiple applications. These can include multiple end-user applications, and reporting and querying tools. So, why even expound on the embedded mode in a book about enterprise development? Well, in order to create a Derby database that is accessible from a networked Derby server, you must first create the database in an embedded mode.

Figure A-1 shows the `ticket` database running as an embedded database in Derby's interactive JDBC scripting tool called ij. Later in this appendix, you will see how to create the `ticket` database using the ij tool. While ij is connected to the `ticket` database as an embedded database, no other applications, processes, or JVMs can access the `ticket` database.

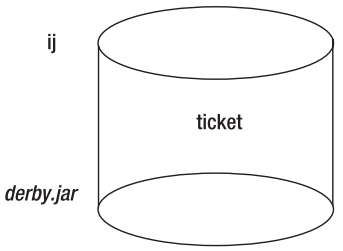

Figure A-1. *ticket database running in the ij process*

The only requirement in order to use Derby as an embedded database is for the `derby.jar` file that comes with Derby to be in the classpath. The embedded connection URL has a format of `jdbc:derby:<databaseName>`. So the `ticket` database connection URL would look like `jdbc:derby:ticket`.

Network Server Mode

Most enterprise applications require the ability to attach to a database from multiple applications, processes, or JVMs by multiple users. Figure A-2 shows the `ticket` database running in the network server mode and being accessed by ij, Eclipse Web Tools, and the Trouble Ticket application.

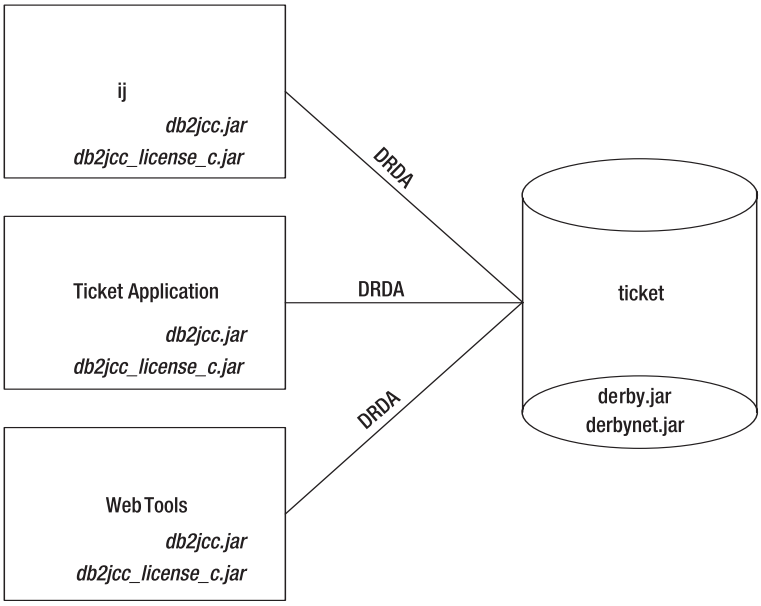

Figure A-2. *ticket database running as a network server and being accessed by multiple applications*

Using Derby in network server mode is a little more complex than using the embedded version, but is far simpler than working with most network databases. To begin, the network server must run as its own process. The easiest way to start the server is with a script or batch file. There are example startup scripts later in the appendix in the section "Running Derby Server." The server process is a Java application and must contain both the `derby.jar` and `derbynet.jar` files in the classpath.

Client applications communicate with Derby using the Open Group's (`http://www.opengroup.org`) open database access standard known as Distributed Relational Database Architecture (DRDA). The DRDA defines a specification for communications between applications and distributed relational databases. Both Derby and Cloudscape use the IBM DB2 JDBC Universal drivers, which are based on the DRDA specification. The drivers are contained in the `db2jcc.jar` and `db2jcc_license_c.jar` files. These files are not included in the Derby distribution and must be downloaded separately from IBM's website at `http://www.ibm.com/developerworks/db2/downloads/jcc/`. The drivers are made available free of charge, but be aware the accompanying license agreement requires the original license agreement be included if the JAR files are distributed with an application. In addition, you will have to register with IBM if you have not already to download the drivers.

The Derby network server URLs differ slightly from embedded server URLs because they must include the host name of the machine running the server. The format is `jdbc:derby:net://<server>[:port]/<databaseName>`. Notice the `net` ensures the driver manager uses the DRDA protocol and drivers rather than the embedded drivers. `server` is the DNS name or the IP address of the server. `port` is the port in which Derby is running. The default port is 1527 and can be eliminated from the connection URL if the server is using the default port. `databaseName` is

the name of the database trying to be accessed. `jdbc:derby:net://localhost/ticket` is an example of a URL used to connect to a `ticket` database of a Derby network server running on the same machine as the client accessing it.

Using Derby

To use Derby as a network server, you must install Derby, create a new database, and run the server.

Installing Derby

To install Derby as a network server, you must download and uncompress the Derby binary distribution and IBM DB2 JDBC Universal drivers. Both are made available as zip files. For those using Unix, Linux, or OS X, Derby is also available as a `tar.gz` file.

■ **Note** Use the latest official release of Derby from `http://incubator.apache.org/derby/derby_downloads.html` or `http://db.apache.org` after Derby moves from the incubator. This example uses Derby 10.0.2.1, the release version available at the time of this writing.

To install Derby, complete the following:

1. Download the `incubating-derby-10.0.2.1-bin.tar.gz` file for Unix, Linux, or OS X, or the `incubating-derby-10.0.2.1-bin.zip` file for Windows from `http://incubator.apache.org/derby/derby_downloads.html`.
2. Uncompress the `incubating-derby-10.0.2.1-bin` file to `<home>/devl/incubating-derby-10.0.2.1-bin` on Unix, Linux, or OS X, or `C:\devl\incubating-derby-10.0.2.1-bin` on Windows.
3. Rename the directory `incubating-derby-10.0.2.1-bin` to `derby-10.0.2.1` to make the directory name shorter and clearer by including only the project name and version number.
4. Download the `db2jcc_for_derby.zip` file from `http://www.ibm.com/developerworks/db2/downloads/jcc/`.
5. Unzip `db2jcc_for_derby.zip` to `<home>/devl/db2jcc` on Unix, Linux, or OS X, or `C:\devl\db2jcc` on Windows.

■ **Note** This installation assumes you already have JDK1.3 or greater installed.

Creating Derby Database

As mentioned earlier, a Derby database must be created in the embedded mode. The easiest way to create the database is to use the ij command-line client included with Derby. The ij client can be executed from the command line but is more conveniently executed from a script or batch file. Unfortunately, Derby does not come with any ij scripts or batch files, so Listing A-1 provides an example of a Bash script for Unix, Linux, or OS X. Listing A-2 is a batch file for Windows.

Listing A-1. *Example of an ij Bash Script*

```
#!/bin/sh

CLASSPATH=lib/derby.jar:lib/derbynet.jar:lib/derbytools.jar:../db2jcc/lib/db2jcc
.jar:../db2jcc/lib/db2jcc_license_c.jar

java -Dij.driver=com.ibm.db2.jcc.DB2Driver -cp $CLASSPATH org.apache.derby.tools
.ij
```

Listing A-2. *Example of an ij.bat File*

```
set CLASSPATH=lib\derby.jar;lib\derbynet.jar;lib\derbytools.jar;
..\db2jcc\lib\db2jcc.jar;..\db2jcc\lib\db2jcc_license_c.jar

java -Dij.driver=com.ibm.db2.jcc.DB2Driver -cp %CLASSPATH% org.apache.derby.tools.ij
```

In Listings A-1 and A-2, the `CLASSPATH` variable is configured to include all the JAR files ij needs to access both embedded and networked Derby databases. The `derby.jar` file is included so ij can access embedded databases. The `derbynet.jar`, `db2jcc.jar`, and `db2jcc_license_c.jar` files are included for accessing network servers. ij and other admin tools like sysinfo and dblook are included in the `derbytools.jar` file.

The next line of Listings A-1 and A-2 executes the JVM, instructing it to run the main class of ij, `org.apache.derby.tools.ij`. Using the system property `ij.driver`, it instructs ij to use the `com.ibm.db2.jcc.DB2Driver` JDBC driver.

Once ij is running, connecting to an embedded database with the attribute `create=true` will create a new database. The new database will exist as a directory in the current working directory with the same name as the database. For example, connecting to `jdbc:derby:ticket;create=true` will create a new Derby `ticket` database.

To create a new `ticket` database, complete the following:

1. If you have not done so, create an ij script or `ij.bat` file like the ones in Listings A-1 and A-2 in the `<home>/devl/derby-10.0.2.1` or `C:\devl\derby-10.0.2.1` directory.
2. Execute the ij script or `ij.bat` file.
3. At the `ij>` prompt, enter **connect 'jdbc:derby:ticket;create=true';**.
4. To quit ij, enter **exit;**.

■ **Warning** You must exit ij after creating the database; otherwise, the server will not have access to the database, since only one JVM has access to an embedded database.

After completing these tasks, you should have a `<home>/devl/derby-10.0.2.1/ticket` directory on Unix, Linux, or OS X, or a `C:\devl\derby-10.0.2.1\ticket` directory on Windows.

■ **Note** ij does not have to be used interactively. ij can also execute commands in a file.

Running Derby Server

In order to start and stop a Derby server, you must execute the `org.apache.derby.drda.NetworkServerControl` class, passing it either a start or shutdown argument. Unfortunately, Derby does not include scripts or batch files for starting or stopping Derby servers. So Listings A-3 and A-4 are examples of startup Bash scripts for Unix, Linux, and OS X, and a batch file for Windows, respectively.

Listing A-3. *Example of Startup Bash Script*

```
#!/bin/sh
CLASSPATH=lib/derby.jar:lib/derbynet.jar
java -cp $CLASSPATH org.apache.derby.drda.NetworkServerControl start
```

Listing A-4. *Example of startup.bat File*

```
set CLASSPATH=lib\derby.jar;lib\derbynet.jar
java -cp %CLASSPATH% org.apache.derby.drda.NetworkServerControl start
```

Listings A-5 and A-6 are examples of a shutdown Bash script for Unix, Linux, and OS X, and a batch file for Windows, respectively.

Listing A-5. *Example of Shutdown Bash Script*

```
#!/bin/sh
CLASSPATH=lib/derby.jar:lib/derbynet.jar
java -cp $CLASSPATH org.apache.derby.drda.NetworkServerControl shutdown
```

Listing A-6. *Example of shutdown.bat File*

```
set CLASSPATH=lib\derby.jar;lib\derbynet.jar
java -cp %CLASSPATH% org.apache.derby.drda.NetworkServerControl shutdown
```

Notice in Listings A-3, A-4, A-5, and A-6 that the scripts and batch files begin by including the `derby.jar` and `derbynet.jar` files in the classpath. These two JAR files contain all the classes necessary to run a Derby server. Both scripts also launch the JVM with a main class of `org.apache.derby.drda.NetworkServerControl`. The only difference between the startup scripts and shutdown scripts is the passing of the `start` argument, which causes a Derby server to start up and listen on port 1527. The shutdown scripts pass an argument of `shutdown`, which stops the Derby server.

■ **Note** On Windows, the startup script cannot be named `start.bat` to match the `start` argument passed to the `NetworkServerControl` class because it conflicts with the `start.exe` program.

To start the Derby server, complete the following:

1. If you have not done so already, create a startup script in the `<home>/derby-10.0.2.1` directory with the contents of Listing A-3 or a `startup.bat` file in the `C:\devl\derby-10.0.2.1` directory with the contents of Listing A-4.
2. Execute the startup script or `startup.bat` file; you should see the message "Server is ready to accept connections on port 1527."

To verify the server is running, use ij to connect to the `ticket` database.

1. Execute the ij script on Unix, Linux, or OS X, or the `ij.bat` file on Windows.
2. At the `ij>` prompt, enter **connect 'jdbc:derby:net://localhost/ticket:user=sa;password=sa;';**.

If the connection is successful, you should return to the `ij>` prompt. You should also see a `Connection number: 1.` in the Derby console. If the connection is not successful, you will see a `java.net.ConnectionException`.

■ **Note** The user name and password can be anything since security is not enabled, but they must be included. Otherwise you will see a "null userid" or "null password not supported" error message.

To stop the Derby server, complete the following:

1. If you have not done so already, create a shutdown script in the `<home>/derby-10.0.2.1` directory with the contents of Listing A-5 or a `shutdown.bat` file in the `C:\devl\derby-10.0.2.1` directory with the contents of Listing A-6.
2. Execute the shutdown script on Unix, Linux, or OS X, or the `shutdown.bat` file on Windows.

Summary

In this appendix, we touch the surface of the popular open source database Derby. We showed you just enough to be able to create a Derby database and run a Derby network server in order to use the examples in this book. If you want more information about Derby, visit the main Derby site, `http://incubator.apache.org/derby/`, or the Derby manuals, `http://incubator.apache.org/derby/manuals/`.

APPENDIX B

JBoss Application Server

J2EE applications like enterprise (EAR), web (WAR), and EJB JARs must run in the context of an application server. There are many commercial and open source applications servers available. The most popular commercial servers include IBM WebSphere (`http://www.ibm.com/software/webservers/appserv/`) and BEA WebLogic (`http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/server/`). The open source servers include ObjectWeb JOnAS (`http://jonas.objectweb.org`) and the focus of this appendix, JBoss Application Server (AS) (`http://www.jboss.org/products/jbossas`). A comprehensive list of J2EE application servers is available at TheServerSide Application Server Matrix (`http://www.theserverside.com/reviews/matrix.tss`).

While WTP should be compatible with any J2EE 1.4–compliant application server, we chose to use JBoss AS for the examples in this book. JBoss AS is an ideal candidate because it is open source, so everybody has access to it. The remainder of this appendix provides an overview of JBoss AS as well as installation and configuration instructions. It concludes by showing how to configure a JNDI reference to the Apache Derby `ticket` database created in Appendix A.

JBoss AS Overview

According to BZ Research, JBoss AS is the leader in the J2EE application server market. During the past couple of years, JBoss AS slowly and silently surpassed WebSphere and WebLogic in popularity and production installs. JBoss AS's popularity is due largely to the following features:

- **Open sourced under the GNU Lesser General Public License (LGPL)**: This enables organizations to deploy and embed the application server for free.
- **J2EE 1.4 certified (JBoss AS version 4.0)**: JBoss AS is the first 1.4-certified application server and the first ever certified open source application server.
- **Service-Oriented Architecture using JMX extensions**: The flexible microkernel architecture is easily extended using standard JMX extensions.
- **Relatively small footprint**: The flexible microkernel architectures enable the application server to be configured to run only necessary services.
- **High performance**: Independent benchmarks show JBoss AS performs as well as other leading application servers.

- **Clustering support**: This enables applications to scale as demand grows.
- **Aspect-Oriented Programming (AOP) model**: This enables you to transparently add service behavior to POJOs.
- **Hot deploy**: Installing or uninstalling applications is as easy as copying files to or deleting files from the `deploy` directory.
- **Community support**: The community answers questions in forums and provides free documentation.
- **Commercial support**: JBoss, Inc. provides support for a fee.

As Chapter 1 mentions, an application server contains both a web container and an EJB container. The JBoss AS is no different. Figure B-1 illustrates the JBoss AS architecture.

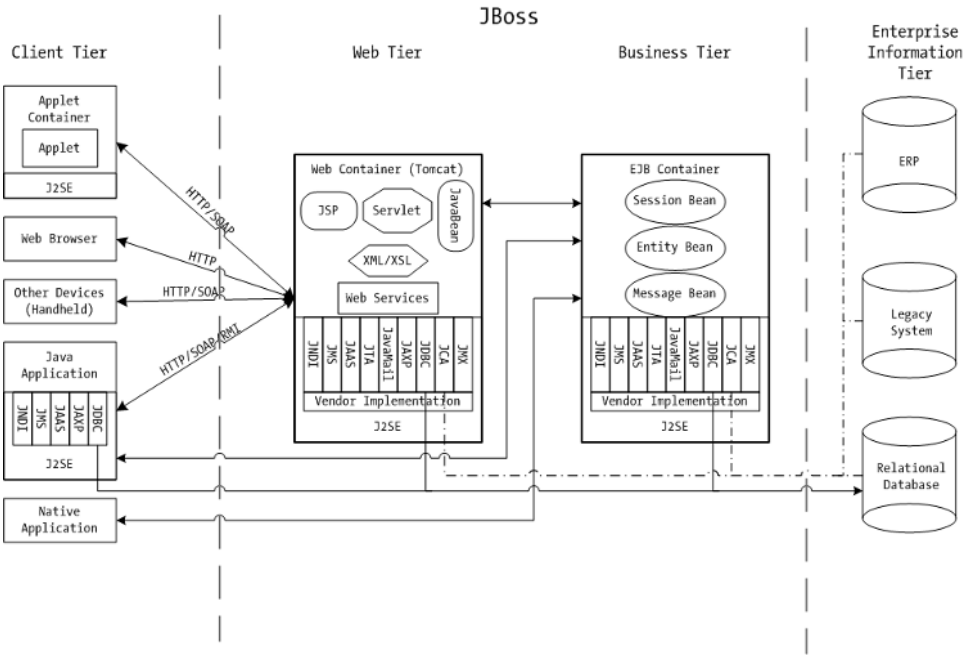

Figure B-1. *JBoss AS architecture*

When you download and install JBoss AS, you get an implementation of a web container and EJB container denoted in Figure B-1 by the vertical dashed lines. JBoss, Inc. and the JBoss community have implemented their own EJB container. However, the popular open source Apache Jakarta Tomcat (`http://jakarta.apache.org/tomcat/`) web container has been chosen to fulfill the web container requirement of the J2EE Specification.

■ **Note** Earlier versions of JBoss AS alternated between the open source web containers of Tomcat and Jetty (`http://jetty.mortbay.org`). This proves the "best of breed" goal identified in the J2EE Specification is obtainable. However, since version 3.2.4, JBoss AS has standardized on Tomcat.

Using JBoss AS

In order to use JBoss AS, you must first install it. After installation, the server must be started using supplied scripts.

Installing JBoss AS

JBoss AS must be the easiest J2EE application server to install. It does not even require running an installation program or script. Just download the binary version bundled as a `tar.bz2` file for Unix, Linux, or OS X, or a zip file for Windows and uncompress it. The JBoss AS does require a JDK 1.4 or higher to be installed and accessible.

■ **Note** Use the latest official release of JBoss AS from `http://www.jboss.org/downloads/index`. This example uses JBoss AS 4.0.1 sp1, the latest release version available at the time of this writing. This version includes Tomcat 5.0 as its web container.

■ **Warning** Make sure the directory path JBoss AS is installed to contains no spaces.

To install JBoss AS, follow these steps:

1. Download the `jboss-4.0.1sp1.tar.bz2` file for Unix, Linux, or OS X, or the `jboss-4.0.1sp1.zip` file for Windows from `http://www.jboss.org/downloads/index` or directly from the SourceForge project at `http://sourceforge.net/projects/jboss/`.
2. Uncompress the `jboss-4.0.1sp1.tar.bz2` or `jboss-4.0.1sp1.zip` file to a local directory such as `~/devl` on Unix, Linux, or OS X, or `C:\devl\` on Windows.

■ **Note** For more information on installing and configuring JBoss AS, see "Getting Started with JBoss 4.0" at `http://docs.jboss.org/jbossas/getting_started/startguide40/` or the JBoss 4 Application Server Guide at `http://docs.jboss.org/jbossas/jboss4guide/r2/html/`.

After the installation, you should have a directory structure similar to the one shown in Figure B-2.

Figure B-2. *JBoss AS directory structure*

In Figure B-2, you can see the root directory is `jboss-4.0.1sp1`. Just beneath that are the `bin`, `client`, `docs`, `lib`, and `server` directories, which contain various items as described here:

- The `bin` directory contains scripts for starting and stopping the JBoss AS as well as other scripts.
- The `client` directory contains the JAR files client applications such as a Swing application that might need to interact with the JBoss AS. The `client` directory also contains a `jbossall-client.jar` file, which encompasses all the other JARs in the client directory to make deployment easier and the client application classpath smaller.
- The `docs` directory contains DTDs and schemas used by the JBoss AS configuration files. The DTDs and schemas are also a great source of documentation regarding configuration options. License of other open source projects used by JBoss AS are also included in the `doc` directory. The `docs/examples` directory contains examples of different resource and service configurations. For instance, there is an Apache Derby configuration example we will discuss in the section "Configuring Derby Datasource" later in this appendix.
- The `lib` directory contains JAR files that belong on the server's classpath. It is intended to only include the JAR files that come with JBoss AS, so you should not add JAR files to this directory. The "Server Configurations" sidebar explains the proper place to put your JAR files and third-party JARs such as datasource drivers.
- The `server` directory contains different server configurations. See the "Server Configurations" sidebar for more information on JBoss AS configurations.

Running JBoss AS

In JBoss AS's `bin` directory, you will find scripts for starting and stopping the JBoss AS process. On Unix, Linux, or OS X, the `run.sh` script will start the default server configuration that corresponds to the configurations found in the `server/default` directory. On Windows, the `run.bat` script does the same thing. Other server configurations can be started by passing the server

configuration name as a parameter to the run script. See the "Server Configurations" sidebar for more information about running alternative server configurations.

The `shutdown.sh` script on Unix, Linux, or OS X, or the `shutdown.bat` script on Windows can be used to stop the server. Alternatively, if JBoss AS was manually started in a console window, you can use Ctrl+C to stop the server.

SERVER CONFIGURATIONS

One of JBoss AS's most impressive features is the ability to scale down as well as up. If you need a lean server with a small memory footprint, JBoss AS can be configured to run only the bare minimum services you require. This is accomplished through *server configurations*. Server configurations are stored in directories under JBoss AS's server directory (see Figure B-2). JBoss AS comes with three preconfigured configurations: minimal, default, and all.

The minimal configuration starts no services except logging. Default, as indicated by the name, is the default configuration and starts all the commonly used services including the Tomcat service, administrative applications, and an embedded Hypersonic database. The all configuration starts every service that comes bundled with JBoss AS; it includes the default configuration services as well as clustering, IIOP, SNMP, and remoting.

Figure B-2 shows the common configuration directories under the default configuration: `conf`, `data`, `deploy`, `lib`. Additional directories of `data`, `log`, `tmp`, and `work` are created when the server is started up for the first time. The `data` directory contains the contents of the embedded Hypersonic database configured in the `deploy/hsqldb-ds.xml` configuration. The `log` directory contains logs for its particular server configuration. The `tmp` directory is a temporary directory the server uses for staging deployments. The `work` directory contains the uncompressed web applications and servlet code generated from JSP pages.

The remaining `conf`, `lib`, and `deploy` directories can be used to configure JBoss AS before and in some cases while the JBoss AS process is running. The `conf` directory contains XML- and property-based configuration files. These files can be used to configure things like logging, the HTTP port, and services started at startup. The `lib` directory contains JARs needed by the specific server configuration. This is the location for third-party JARs such as database drivers. If you have your own JARs that are shared across applications, this would be the place to put them, assuming they are not already included in a WAR, EAR, or EJB JAR.

■ **Warning** Classes in the server configuration `lib` directory are shared across all applications in this configuration. If you need to update the JAR, you will have to restart the entire server. In addition, static values will be shared across all applications.

The `deploy` directory is unique because changes here happen real time while the JBoss AS process is running. To install an application, simply copy the EAR, WAR, or EJB JAR file into this directory and it will be immediately loaded and configured. Remove the EAR, WAR, or EJB JAR file, and the application will be unloaded. Replace the EAR, WAR, or EJB JAR file, and the application will be hot-swapped by unloading the running application and all of its classes and automatically load the new application. It can also be used to start new services and/or new configurations like pooled datasources.

To create a new custom configuration, simply copy one of the existing server configurations. Then add or remove the appropriate services. One approach for the example Trouble Ticket application would be to make a copy of the default server configuration and name it `ticket`. This is similar to the concept of domains that some other application servers like WebLogic have. The goal would be to have a named configuration with all the application or related application's configurations organized together. This also preserves the original default configuration if you need to get back to it.

To execute a server configuration other than the default configuration, execute the run script, passing the name of the server configuration to run. For example, to run the all configuration on Unix, Linux, or OS X, enter **./run.sh all**, or on Windows enter **run all**.

To start the default server configuration, follow these steps:

1. Open a command prompt and change directories to JBoss AS's `bin` directory. For example, on Unix, Linux, or OS X, use `cd ~/devl/jboss-4.0.1sp1/bin`; on Windows, use `cd C:\devl\jboss-4.0.1sp1\bin`.
2. Execute the run script by entering **./run.sh** on Unix, Linux, or OS X, or entering **run** on Windows or double-clicking the `run.bat` file in Windows Explorer.
3. When the server has finished the startup process, you should see a startup message similar to `[Server] JBoss (MX MicroKernel) [4.0.1sp1 (build: CVSTag=JBoss_4_0_1_SP1 date=200502160314)] Started in 20s:489ms` in the console window.

JBoss AS contains some web-based administrative applications including a Tomcat status page, a JMX console, and a JBoss web console. These applications are automatically started when JBoss AS's default server configuration is run. You can visit these applications to ensure the server has started properly.

To verify JBoss AS started correctly, follow these steps:

1. Open a web browser.
2. Navigate to `http://localhost:8080/` if the server is running on the same machine as your browser. Otherwise, substitute your server name or IP address for `localhost`.
3. Verify that the resulting web page looks like the one in Figure B-3.

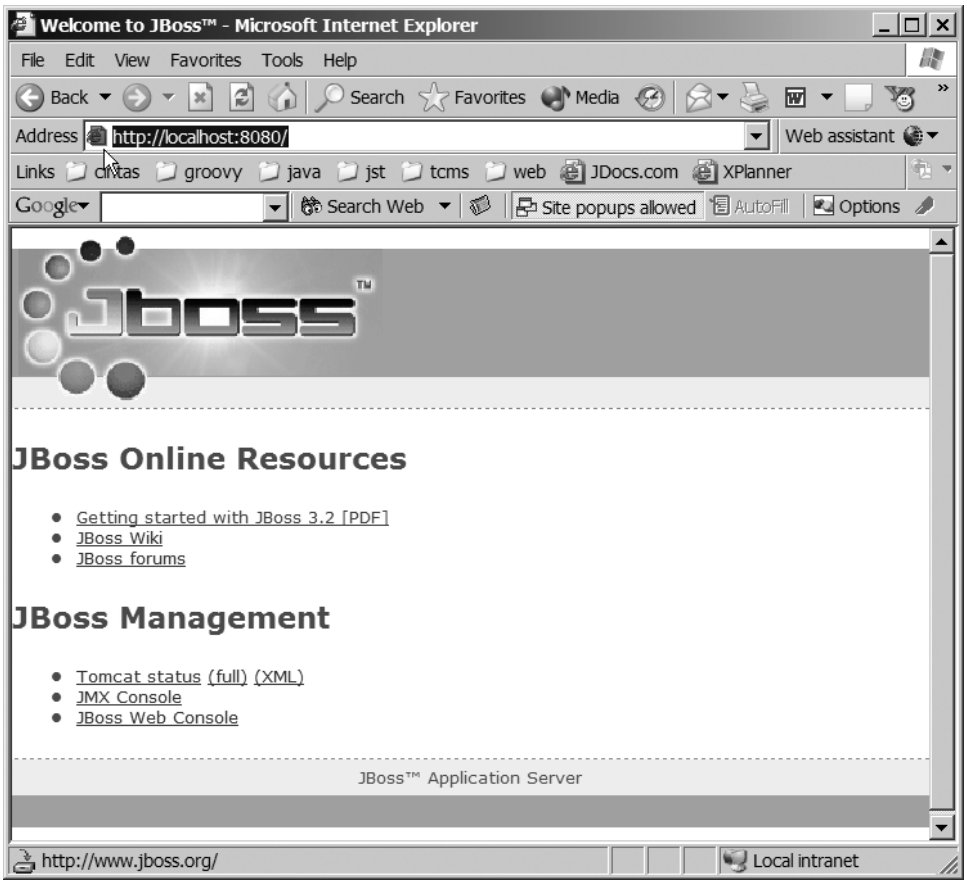

Figure B-3. *JBoss AS administrative applications*

Configuring Derby Datasource

A key function of J2EE applications servers is to manage resources. Datasource connections are one of the primary resources you should let the application server manage. The application server will automatically handle difficult issues like *connection pooling*. Connection pooling enables multiple applications to share datasource connections in order to reduce the expense of establishing new connections for each request. In addition, application servers usually provide tools for administrators to monitor the number of connections being used and provide the ability to add new connections to the pool at runtime if necessary.

In order to use a datasource, you must first configure it. The process of configuring a datasource is application server–specific. Many have a web-based administrative console for configuring the datasource. JBoss AS, however, uses an XML-based configuration file that can be dropped into the server configuration `deploy` directory. This can even be done while the application server is running. JBoss AS will recognize the new configuration file and automatically configure the datasource. You can find examples of configuration files for most common

databases in JBoss AS's `docs/examples/jca` directory on Unix, Linux, or OS X, and `docs\examples\jca` on Windows. The file names include the database name followed by the required `-ds.xml` extension.

■ **Note** JBoss AS does include a `derby-ds.xml` file to demonstrate an Apache Derby configuration. Unfortunately, the example is tailored to configuring an embedded Derby database and not a network server like the one set up in Appendix A.

A JBoss AS datasource configuration file contains the standard JDBC configuration information to connect to a database such as the connection URL, driver class, user name, and password. It also contains a JNDI name that can be used by an application to look up the datasource. The JNDI name may be referenced in deployment descriptors for container-managed entity beans or directly in the source code itself. The configuration file can also contain instructions for the application server such as pool sizes and time outs. Listing B-1 shows an example of a configuration file for the Apache Derby database configured in Appendix A.

■ **Note** We recommend naming the datasource configuration file the name of your database followed by a `-ds.xml`. For example, the `ticket` database would be `ticket-ds.xml`.

Listing B-1. *Appendix A's Apache Derby Datasource Configuration*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Pro Eclipse JST Plug-in Ticket DB data source -->
<datasources>
  <local-tx-datasource>
    <jndi-name>jdbc/ticket</jndi-name>
    <connection-url>jdbc:derby:net://localhost:1527/ticket</connection-url>
    <driver-class>com.ibm.db2.jcc.DB2Driver</driver-class>
    <user-name>sa</user-name>
    <password>sa</password>

    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>5</idle-timeout-minutes>
    <track-statements>true</track-statements>
  </local-tx-datasource>
</datasources>
```

The example in Listing B-1 shows the root element of a datasource configuration file is a `datasources` element. Contained within the `datasources` element is a block that describes the specifics of the connection, with the parent element determining the connection type. Connection types can determine whether the datasource supports distributed transactions

(XADataSource), nondistributed transactions (non-XADatasource), or no transactions. In this case, it identifies this connection is a non-XADatasource with local transactions. Next is the JNDI name. This example uses a JNDI name of `jdbc/ticket`. The datasource name `java:/jdbc/ticket` can be used to look up a pooled datasource connection in either source code or deployment descriptors. The `connection-url`, `driver-class`, `user-name`, and `password` values are all discussed in Appendix A.

The `min-pool-size` of 5 instructs JBoss AS to immediately create 5 connections to the datasource. This happens during the initial configuration and/or restart of the server. The configuration limits the number of connections that can be created to 20. As an application scales or during heavy traffic, the application server may create additional connections if the existing connections are already being utilized. `max-pool-size` limits this number. `max-pool-size` can also be useful when databases have connection limits due to licensing. So connections are not left open indefinitely, `idle-timeout-minute` can be configured to close the connections. Of course, it will only close connections until it reaches the minimum pool size.

The last element is `track-statements`. `track-statements` is a debugging technique designed to warn when JDBC code does not close statements or result sets. Not closing statements and result sets leads to connection leaks and ultimately running out of connections in the pool when the max size is reached. It is recommended that `track-statements` be turned off for production.

■ **Note** For more information on the definitions and possible datasource configurations, see `http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch7.chapt.html#ch7.jdbc.sect` or the `jboss-ds_1_5.dtd` file in JBoss AS's `docs\dtd` directory.

JBoss AS will also need access to the datasource drivers. The drivers should be placed in the server configuration's `lib` directory. Adding the drivers to the `lib` directory will require the server process to be restarted in order for JBoss AS to find the appropriate classes.

To configure a datasource for the Apache Derby database set up in Appendix A, follow these steps:

1. Copy the `db2jcc.jar` and `db2jcc_license_c.jar` files to the `~/devl/jboss-4.0.1sp1/server/default/lib` directory on Unix, Linux, or OS X, or the `C:\devl\jboss-4.0.1sp1\server\default\lib` directory on Windows.

2. If JBoss AS is currently running, restart it.

3. Create a JBoss AS datasource configuration file in a temporary location with the contents of Listing B-1 and a name of `ticket-ds.xml`.

4. Copy or move `ticket-ds.xml` to the `~devl/jboss-4.0.1sp1/server/default/deploy` directory on Unix, Linux, or OS X, or the `C:\devl\jboss-4.0.1sp1\server\default\deploy` directory on Windows.

5. In the JBoss AS console, you should see an information message indicating the datasource was bound to the configured JNDI name. For example: `INFO [WrapperDataSourceService] Bound connection factory for resource adapter for ConnectionManager 'jboss.jca:name=jdbc/ticket,service=DataSource Binding to JNDI name 'java:jdbc/ticket'.`

To validate the datasource was also added, you can use JBoss AS's JMX Management Console. Through this console, you can interrogate any JMX bean running in JBoss AS. To see whether the datasource is available, you need to use the JNDIView service. This service has a list method that returns a tree of JNDI names. Included in the list should be the new `jdbc/ticket` name. Figure B-4 shows the `jdbc/ticket` name in the JNDI View.

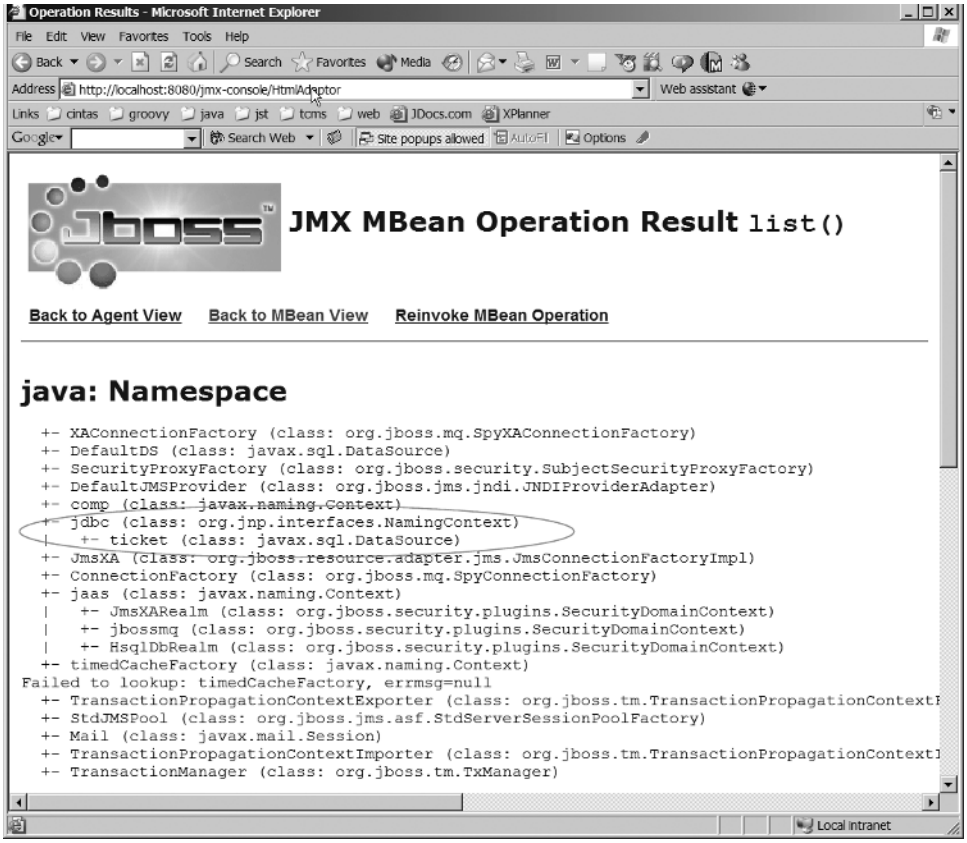

Figure B-4. *Apache Derby datasource listed in the JNDI View*

To verify the JNDI name is bound, follow these steps:

1. Open a web browser.
2. Navigate to `http://localhost:8080/jmx-console/`. This is the same JMX console application available from the administration page you loaded earlier to ensure JBoss AS was running.
3. Select `service=JNDIView` under the section jboss.
4. Click the Invoke button under `java.lang.String list()` (see Figure B-5).

5. Locate the ticket datasource name in the java: Namespace section. It should look similar to what we showed previously in Figure B-4.

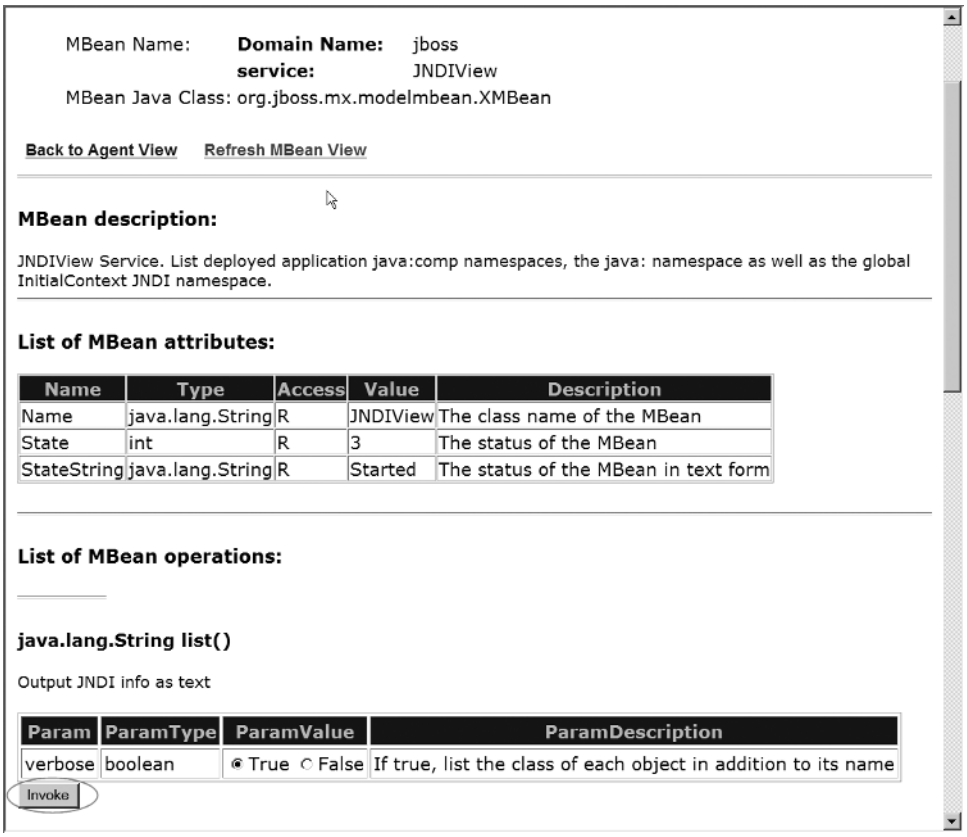

Figure B-5. *JNDI info Invoke button*

Summary

This appendix concludes the process of setting up a development environment that includes an Apache Derby database and a JBoss AS. It explains how to install and configure the JBoss AS. It also included instructions on how to configure a datasource for the database installed in Appendix A.

Index

D

P

Q

R

S

X