



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.

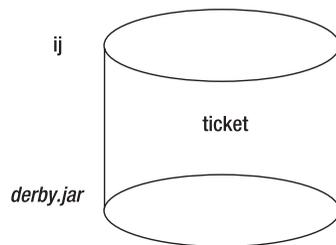


Figure A-1. *ticket* database running in the *ij* process

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.

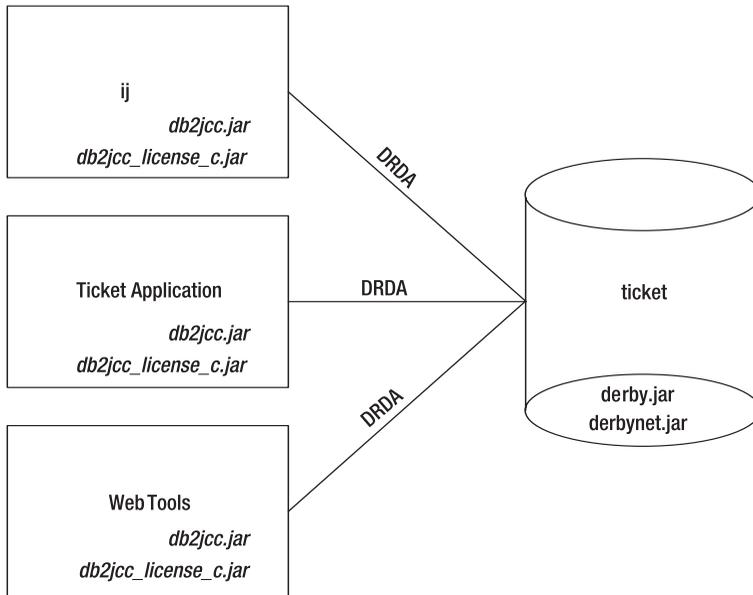


Figure A-2. *ticket database running as a network server and being accessed by multiple applications*

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>/dev1/incubating-derby-10.0.2.1-bin` on Unix, Linux, or OS X, or `C:\dev1\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>/dev1/db2jcc` on Unix, Linux, or OS X, or `C:\dev1\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>/dev1/derby-10.0.2.1` or `C:\dev1\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>/dev1/derby-10.0.2.1/ticket` directory on Unix, Linux, or OS X, or a `C:\dev1\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:\dev\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:\dev\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/>.



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/webervers/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.

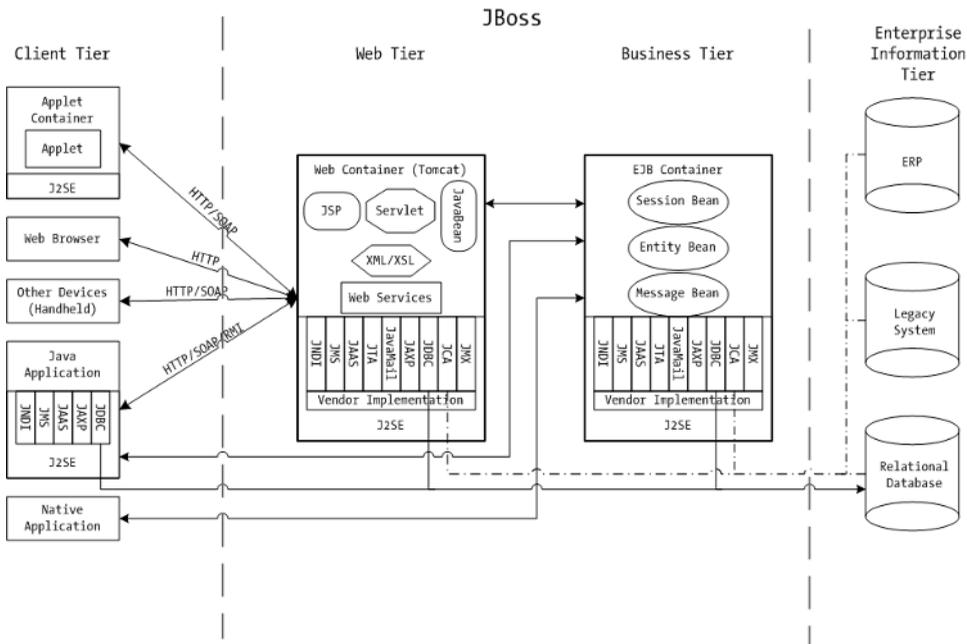


Figure B-1. *JBoss AS architecture*

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 `~/dev1` on Unix, Linux, or OS X, or `C:\dev1\` 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 `docs` 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 ~/dev1/jboss-4.0.1sp1/bin`; on Windows, use `cd C:\dev1\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.

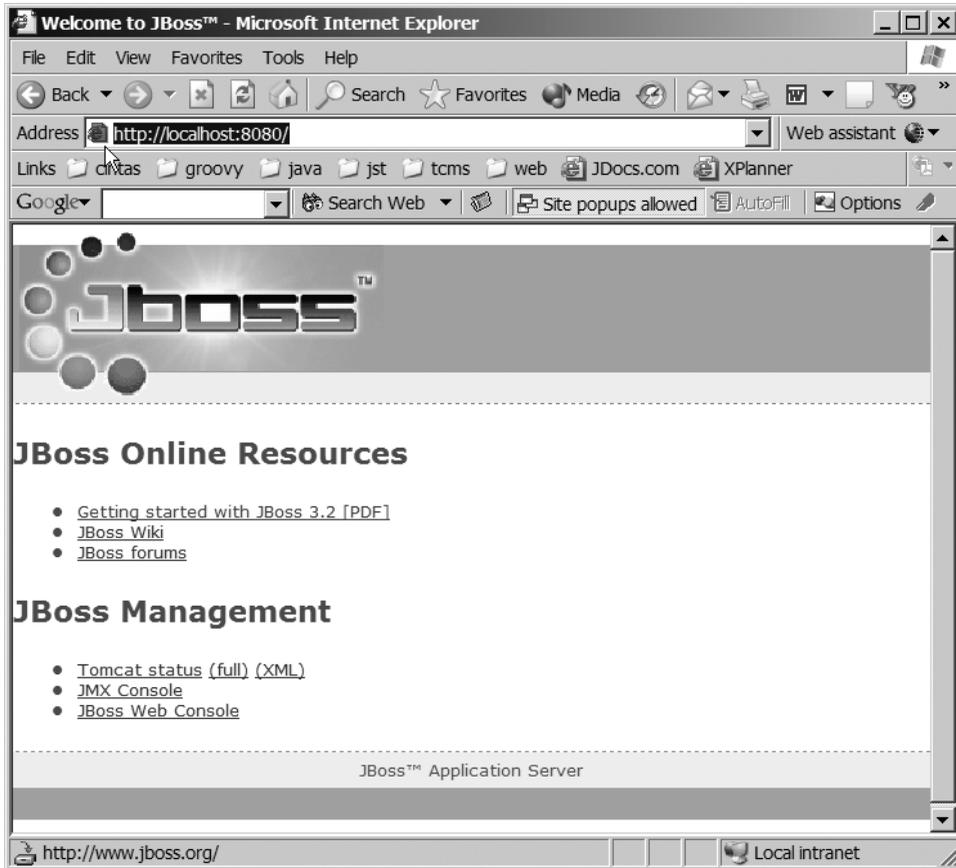


Figure B-3. JBoss AS administrative applications

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 `jdbc-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 `~/dev1/jboss-4.0.1sp1/server/default/lib` directory on Unix, Linux, or OS X, or the `C:\dev1\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 `~/dev1/jboss-4.0.1sp1/server/default/deploy` directory on Unix, Linux, or OS X, or the `C:\dev1\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.

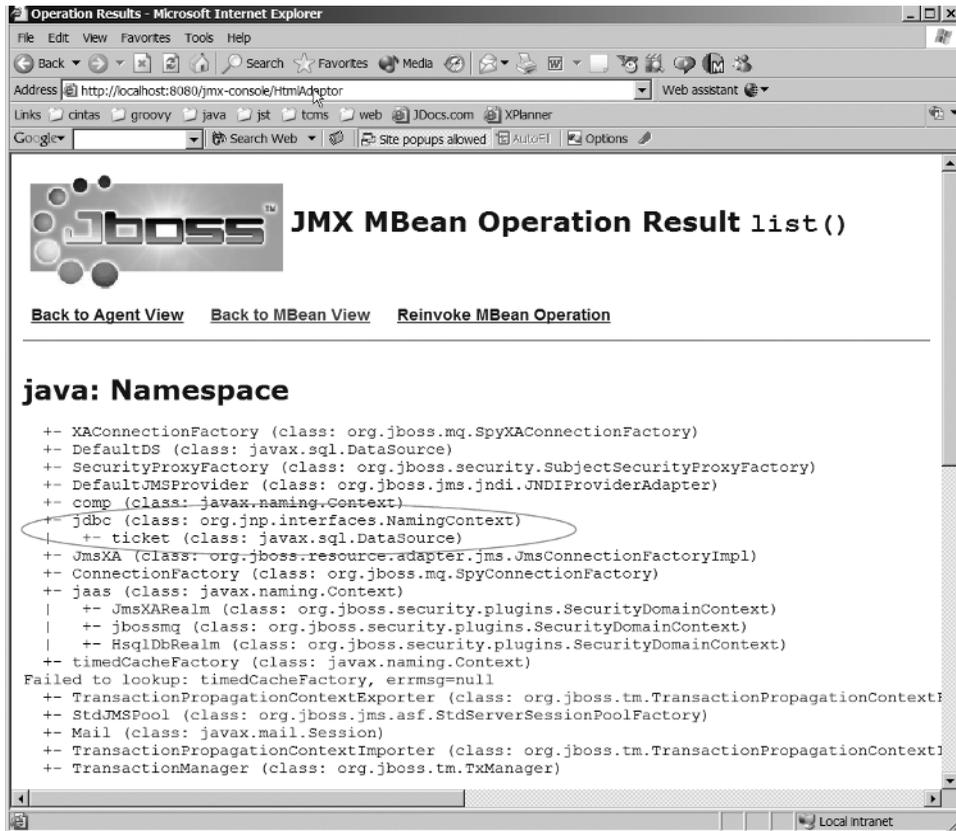


Figure B-4. Apache Derby datasource listed in the JNDI View

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.

MBean Name: **Domain Name:** jboss
service: JNDIView
MBean Java Class: org.jboss.mx.modelmbean.XMBean

[Back to Agent View](#) [Refresh MBean View](#)

MBean description:

JNDIView Service. List deployed application java:comp namespaces, the java: namespace as well as the global InitialContext JNDI namespace.

List of MBean attributes:

Name	Type	Access	Value	Description
Name	java.lang.String	R	JNDIView	The class name of the MBean
State	int	R	3	The status of the MBean
StateString	java.lang.String	R	Started	The status of the MBean in text form

List of MBean operations:

java.lang.String list()

Output JNDI info as text

Param	ParamType	ParamValue	ParamDescription
verbose	boolean	<input checked="" type="radio"/> True <input type="radio"/> False	If true, list the class of each object in addition to its name

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

■ Symbols

- `%oCE`
 - supported by servlets, 221

■ A

- Action class
 - execute method, 205
 - extending, 206
 - `org.apache.struts.action` package, 205
 - overriding execute method, 208–210
- ActionForm class
 - compared to `JavaBean`, 201
 - `org.apache.struts.action` package, 205
- ActionMapping class
 - using, 210
- Actions pane
 - Web Services Explorer, 287
- ActionServlet
 - configuring in `web.xml` file, 203
 - controller for Struts, 201
- American National Standards Institute (ANSI), 47
- annotations overview, 91, 93
- Another Neat Tool. *See* Ant
- ANSI (American National Standards Institute)
 - supported by WST, 47
- Ant
 - web application deployment, 251
 - alternative configuration, 257
 - `build.xml`, 254
 - integration with Eclipse, 254–256
 - working with Ant buildfiles, 258–259
 - Workspace layer, 13
- Apache Derby. *See* Derby
- App Client JAR file
 - importing projects, 83
- applet components, 6
- applet containers, 8
- application client containers, 8
- Application Client Project, 74
 - supported by J2EE Project Tools, 29
- application components, 6–7
- application deployment, 9
- Application Server Matrix
 - TheServerSide.com, 20
- `application.xml` deployment descriptor, 194
 - contents of, 194

- `<assembly-descriptor>` tag, 185
- asynchronous calls supported by JMS, 155

■ B

- bean-managed persistence
 - entity bean persistence mechanism, 120
- Beginning J2EE 1.4: From Novice to Professional
 - James L. Weaver, Kevin Mukhar, and Jim Crume, 162, 266
- `bin` directory
 - installing JBoss application server, 314
- binary data
 - handled by `ServletOutputStream` object, 225
- `<binding>` tag, 280
- BMP (bean-managed persistence), 120
- `build-user.xml` file, 258–259
- `build.xml` file, 252–253, 258
 - properties, 254
 - targets, 253
 - types, 254
- building and deploying web applications, 249
- Business Tier
 - J2EE architecture, 3
- `BytesMessage` interface, 158

■ C

- Cache Resolver
 - WST Internet Tools, 55
- CGI (Common Gateway Interface)
 - and servlets, 218
- `ClassNotFoundException` class, 207
- `.classpath` property
 - configurable properties for J2EE projects, 86
- client directory
 - installing JBoss application server, 314
- Client Environment Configuration page
 - Web Service Client wizard, 282
- Client Tier
 - J2EE architecture, 3
- Cloudscape
 - as forerunner of Derby, 303
- CMP (container-managed persistence), 120
- CMP class, generating, 136–138
- Common Gateway Interface (CGI), 218

- Common Object Request Broker
 - Architecture (CORBA), 264
 - Connection class
 - createSession method, 159
 - Connection wizard, 289–290
 - ConnectionFactory interface, 159
 - createConnection method, 159
 - Connector Project, 74
 - creating from a RAR file, 83
 - supported by J2EE Project Tools, 29
 - containers, introduction, 7–8
 - <copy> tag, 253
 - CORBA (Common Object Request Broker Architecture), 264
 - course-grain access, 121
 - Create Servlet wizard, 32–34
 - createConnection method
 - ConnectionFactory interface, 159
 - createConsumer method
 - Session class, 159
 - createProducer method
 - Session class, 159
 - createSession method
 - Connection class, 159
 - createTopicConnection method
 - TopicConnectionFactory interface, 160
 - Crume, Jim, Mukhar, Kevin, and Weaver, James L.
 - Beginning J2EE 1.4: From Novice to Professional, 162, 266
 - custom RDB perspective, creating, 290–291
- D**
- Data Manipulation Language (DML), 289
 - Data Output View, 63–64, 289
 - results of executing SQL statements, 290
 - data projects, creating, 301–302
 - Data Transfer Object. *See* DTO
 - Database Definition Language (DDL), 289
 - Database Explorer, 63, 289
 - creating new connections, 291
 - creating RDB perspective, 291
 - managing connection, 290
 - datasources element, 318
 - DCOM (Distributed Component Object Model), 264
 - DDL (Database Definition Language), 289
 - Debug component
 - Eclipse Platform Runtime layer, 14
 - decoupling supported by JMS, 155
 - DELETE
 - supported by servlets, 221
 - .deployables directory, 86
 - deploy directory
 - JBoss application server, 315
 - deploying application components, 8
 - deploying EJBs, 185
 - accessing deployed EJBs, 190
 - building EJBs, 186
 - creating server configurations, 187–190
 - publishing EJBs, 186
 - deploying J2EE applications, 9
 - deploying web applications, 249–251
 - managing deployment with Ant, 251–259
 - deployment descriptors, 9
 - application.xml, 194
 - ejb-jar.xml file, 178–179, 181, 182–185
 - generating automatically with XDoclet, 182
 - generating with Eclipse, 182
 - generating with the Servlet wizard, 232
 - placing in WEB-INF directory, 243
 - server-specific web application
 - deployment descriptors, 244, 280
 - writing entity beans with
 - EnterpriseJavaBean wizard, 125
 - writing MDBs with EnterpriseJavaBean wizard, 165
 - writing session beans with
 - EnterpriseJavaBean wizard, 96
 - deployment modules, types, 9
 - Derby, 289, 303
 - architecture, 304
 - embedded mode, 304
 - network server mode, 304–305
 - configuring datasource, 317–320
 - creating connection to ticket database, 294–295
 - creating Derby database, 307–308
 - Derby Embedded JDBC Driver, 293
 - history, 303
 - installing, 306
 - running Derby server, 308–309
 - <description> tag, 185, 194
 - destroy method
 - HttpServlet class, 227
 - <display-name> tag, 185, 194
 - Distributed Component Object Model (DCOM), 264
 - DML (Data Manipulation Language), 289
 - docs directory
 - installing JBoss application server, 314
 - DTD (Document Type Definition)
 - Eclipse support for, 58
 - DTO (Data Transfer Object)
 - converting to entity bean and back, 130, 134
 - generating with XDoclet, 140–146
 - DTO pattern, 90
 - durable topics, 157
 - Dynamic Web Projects, 49, 74
 - configuring, 202
 - deploying web applications, 241

- importing projects as WAR files, 83
 - package deployment module as WAR file, 248
 - Struts, 203–205
 - supported by J2EE Project Tools, 29
 - tag libraries, 205–206
- E**
- EAR files, 192–193
 - contents, 194
 - packaging EJB Project into, 178
 - packaging server-side deployment, 177
 - packaging web applications, 248–249
 - <echo> tag, 253
 - Eclipse
 - architecture, 11
 - Eclipse Platform Runtime layer, 13
 - extension points, 16–17
 - Java 2 Platform layer, 12
 - platform components, 14
 - Workbench layer, 13–14
 - Workspace layer, 13
 - build types, 65–66
 - deployment descriptors generated, 182
 - Eclipse SDK, 14
 - JDT (Java Developers Toolkit), 15–16
 - obtaining, 16
 - PDE (Plug-in Development Environment), 16
 - expansion of user base as goal of WST, 46
 - independent plug-ins, 17
 - installing Eclipse 3.1
 - installing Eclipse SDK 3.1, 67
 - installing JDK, 66
 - lack of critical features, 19
 - obtaining Eclipse Platform, 14
 - plug-in paradigm, 11
 - project types, 74
 - Eclipse Class wizard
 - writing an action, 206
 - Eclipse File wizard, 204
 - Eclipse Import wizard, 82
 - Eclipse Modeling Framework. *See* EMF
 - Eclipse SDK, obtaining, 16
 - Eclipse standard Class page
 - EnterpriseJavaBean wizard, 98, 127, 166
 - Eclipse Web Tools Platform. *See* WTP
 - ECMA (European Computer Manufacturers Association)
 - supported by WST, 47
 - editor model
 - WCM (Web Core Model), 47
 - EJB (Enterprise Java Beans)
 - version 2.1, 4
 - EJB Creation wizard, 38–39
 - EJB JAR file
 - archiving EJBs, 178
 - ejb-jar.xml file, 182–185
 - importing projects as EJB Projects, 83
 - introduction, 178, 180–182
 - packaging EJB modules, 177
 - EJB modules
 - deployment modules, 9
 - EJB JAR file packaging, 177
 - EJB Project
 - supported by J2EE Project Tools, 29
 - EJB Project Creation dialog box
 - packaging EJB Project into an EAR file, 178
 - EJB Projects, 74
 - importing, 83
 - EJB Tools, 38–39
 - <ejb-class> tag, 185
 - ejb-jar.xml file, 178–179, 181, 182–185
 - generating with XDoclet, 108–110, 146–149, 171
 - <ejb-name> tag, 185
 - EJBHome interface
 - javax.ejb package, 90
 - EJBLocalHome interface
 - javax.ejb package, 90, 121
 - EJBLocalObject interface
 - javax.ejb package, 121
 - EJBObject interface
 - javax.ejb package, 90, 266
 - ejbRemove method
 - MessageDrivenBean interface, 162, 169
 - EJBs (Enterprise JavaBeans)
 - components, introduction, 7
 - containers, 8
 - creating EJB clients, 191–192
 - deploying, 177, 185
 - accessing deployed EJBs, 190
 - building EJBs, 186
 - creating server configurations, 187–190
 - publishing EJBs, 186
 - EAR files, 192–194
 - exposing methods as Web Services, 99
 - packaging, 177–179
 - exporting files, 180
 - session beans, 89
 - EMF (Eclipse Modeling Framework), 65
 - installing, 69
 - JET (Java Emitter Templates), 94, 123
 - enterprise application architecture, 1–2
 - Enterprise Application Project, 74
 - deploying web applications, 241
 - supported by J2EE Project Tools, 29
 - Enterprise Archive files. *See* EAR files
 - Enterprise JavaBeans. *See* EJBs
 - <enterprise-beans> tag, 185

- EnterpriseJavaBean wizard
 - writing entity beans, 122–128
 - writing session bean, 93, 104
- entity beans, 119
 - adding functionality, 130–134
 - converting a session bean to entity bean, 128–130
 - generating entity files, 134–150
 - introduction, 7
 - overview, 119–122
 - using from a session bean, 152
 - writing, 122
 - EnterpriseJavaBean wizard, 122–128
 - XDoclet entity bean annotations, 151–152
- entity life cycle methods
 - SQL statement mapping, 121
- <entity> tag, 185
- EntityBean interface
 - javax.ejb package, 127, 130
- error-pages.xml file, 203
- European Computer Manufacturers Association (ECMA), 47
- execute method
 - Action class, 205
- Export wizard
 - exporting projects, 83–84
- exporting artifacts
 - packaging web applications, 249

F

- filter-mappings.xml file, 203
- filters, introduction, 6
- filters.xml file, 203
- fine-grain access, 121
- <form> tag, 222–223

G

- GEF (Graphical Editor Framework), 65
 - installing, 70
- GenericServlet class
 - getInitParameter() method, 229
 - implements Servlet interface, 219
- GET
 - supported by servlets, 221
 - using, 227
- getOutputStream() method
 - HttpServletResponse class, 225
- getWriter() method
 - HttpServletResponse class, 225
- Graphical Editor Framework. *See* GEF

H

- HEAD, supported by servlets, 221
- Help component
 - Eclipse Platform Runtime layer, 14

- home interface
 - generating with XDoclet, 102
- HTML documents
 - creating and editing with WST, 221
- HTML Source Page Editor, 60
- HTTP request types supported by
 - HttpServlet class, 221
- HttpServlet class
 - HTTP request types supported, 221
 - inherits from GenericServlet class, 219
 - servlet life cycle methods, 227
- HttpServletRequest class
 - javax.servlet.http package, 218
 - relationship with Servlet interface, 219
- HttpServletResponse class
 - getWriter() method, 225
 - javax.servlet.http package, 218
 - providing responses, 225–227

I

- IBM DB2, 289
- <icon> tag, 194
- IETF (Internet Engineering Task Force)
 - supported by WST, 47
- Import wizard
 - importing projects, 82
- init method
 - HttpServlet class, 227
- InitialContext class
 - javax.naming package, 108, 146
- <input> tag, 222–223
- Integer class
 - java.lang package, 120
- integration
 - benefits of J2EE platform, 4
 - supported by JMS, 155
- integration version
 - Eclipse build types, 65
- Internet Engineering Task Force (IETF), 47
- Internet Tools, 52
 - Cache Resolver, 55
 - proxy settings, 52–53
 - TCP/IP monitoring, 54–55
 - web browser, 56

J

- J2EE Artifacts Model, 28
- J2EE Connector Architecture
 - version 1.5, 5
- J2EE Core Model. *See* JCM
- J2EE Deployment API
 - version 1.1, 5
- J2EE Editor Model, 28
- J2EE Management API
 - version 1.0, 5
- J2EE Module Tools, 29, 31

- J2EE Perspective
 - access to navigation functionalities, 41
 - introduction, 41–42
 - invoking EnterpriseJavaBean wizard, 93
- J2EE Project Model, 28
- J2EE Project Tools
 - projects supported, 29, 31
- J2EE projects
 - creating, 75
 - creating J2EE Runtime Environment with Server Creation wizard, 77–79
 - exporting projects, 83–84
 - importing projects, 82
 - J2EE Runtime Library, 79
 - New Project wizard, 80–81
 - project folder contents, 85
 - .deployables directory, 86
 - project properties, 84–85
 - working with projects, 87–88
- J2EE Runtime Environment
 - creating with Server Creation wizard, 77–79
 - required for J2EE projects, 75
- J2EE SDK
 - Cloudscape bundled with, 303
- J2EE Server Model, 28
- J2EE Server Tools
 - introduction, 31–32
- J2EE specification, 1
 - application components, 6–7
 - application deployment, 9
 - architecture, 2
 - containers, 7–8
 - deployment, 8
 - enterprise application architecture, 1–2
 - overview, 3
 - platform, 4
 - APIs, 4–5
 - services, 8
- J2EE Standard Tools. *See* JST
- J2SE APIs
 - available to J2EE applications, 5
- JAAS (Java Authentication and Authorization Service)
 - supported by JST, 27
- JACC (Java Authorization Service Provider Contract for Containers)
 - version 1.0, 5
- JAR files
 - compiled classes packaged into, 244
- Java 2 Platform layer, 12
- Java API for XML Processing (JAXP), 5
- Java API for XML Registries (JAXR), 5
- Java API for XML-based RPC (JAX-RPC), 5
- Java Authentication and Authorization Service (JAAS), 27
- Java Authorization Service Provider Contract for Containers (JACC), 5
- Java Bean Web Service
 - creating, 268
 - Object Selection Page, 276
- Java Data Objects (JDO), 119
- Java Developers Toolkit (JDT), 15–16
- Java Development Kit (JDK), 66
- Java Edit Model. *See* JEM
- Java Emitter Templates. *See* JET
- Java Management Extensions (JMX), 5
- Java Message Service. *See* JMS
- Java modules
 - deployment modules, 9
- Java Naming and Directory Interface. *See* JNDI
- Java Runtime Environment. *See* JRE
- Java Transaction API. *See* JTA
- Java Web Services Tools, 40, 61–62
- java.lang package
 - Integer class, 120
 - Object class, 169
 - String class, 120
- java.rmi package
 - RemoteException class, 90
- JavaBeans Activation Framework
 - version 1.0, 5
- <javac> tag, 253
- JavaMail
 - services provided, 8
 - version 1.3, 5
- JavaServer Pages. *See* JSP
- javax.ejb package
 - EJBHome interface, 90
 - EJBLocalHome interface, 90, 121
 - EJBLocalObject interface, 121
 - EJBObject interface, 90, 266
 - EntityBean interface, 127, 130
 - LocalObject interface, 90
 - MessageDrivenBean interface, 160–161, 166, 169
 - MessageDrivenContext interface, 162
 - SessionBean interface, 127, 130
- javax.jms package
 - Message interface, 161
 - MessageListener interface, 160, 166, 169
- javax.naming package
 - InitialContext class, 108, 146
 - NamingException class, 97, 126
 - NoInitialContextException class, 118
- javax.servlet package
 - Servlet interface, 219
 - ServletRequest interface, 218
 - ServletResponse interface, 218
- javax.servlet.http package
 - HttpServletRequest class, 218–219
 - HttpServletResponse class, 218

- jaws.xml deployment descriptor, 182
- JAX-RPC (Java API for XML-based RPC)
 - version 1.1, 5
- JAXP (Java API for XML Processing)
 - version 1.2, 5
- JAXR (Java API for XML Registries)
 - version 1.0, 5
- JBoss
 - application server, 311
 - configuring Derby datasource, 317–320
 - installing, 313–314
 - overview, 311–312
 - running, 314
 - server configurations, 315–316
 - configuring topics and queues, 173
 - installation and management of server, 188
 - installing, 71–72
- jboss.xml deployment descriptor, 182
- jboss.xml file
 - generating with XDoclet, 110–111, 171
- jbosscmp-jdbc.xml deployment descriptor, 182
- JCM (J2EE Core Model), 27
 - introduction, 28
- JCP standards
 - technologies supported by JST, 26
- JDBC services provided, 8
- JDK (Java Development Kit)
 - installing, 66
- JDO (Java Data Objects)
 - persistence specification, 119
- JDT (Java Developers Toolkit), 15–16
- JEM (Java Edit Model), 65
 - installing, 70
- JET (Java Emitter Templates), 94, 123
- JFace Workbench layer, 14
- JFreeChart library
 - chart-generation library, 233
- JMS (Java Message Service)
 - Java's implementation of MOM, 155
 - overview, 156–157
 - JMS API, 158–160
 - JMS implementations, 160
 - messages, 157–158
 - services provided, 8
 - version 1.1, 5
 - writing a JMS producer, 174–175
- JMS API, 158–160
- JMX (Java Management Extensions)
 - version 1.2, 5
- JMX Management Console
 - JBoss application server, 320
- JNDI services provided, 8
- JNDI (Java Naming and Directory Interface), 265
 - supported by JST, 27
- jonas-ejb-jar.xml deployment descriptor, 182
- JRE (Java Runtime Environment)
 - creating with JRE wizard, 75–77
 - required for Eclipse SDK 3.1, 66
 - required for J2EE projects, 75
- JRE wizard
 - creating JRE (Java Runtime Environment), 75–77
- JSP (JavaServer Pages), 197
 - introduction, 6
 - JSP Standard Tag Library, 200
 - library categories, 200
 - overview, 198–200
 - servlets compared to, 217
 - Struts overview, 201
 - version 2.0, 5
 - writing pages, 211
 - JSP wizard, 211–213, 215
- JSP editor
 - as SSE (Structured Source Editor), 213
- JSP Standard Tag Library, 200
 - library categories, 200
- JSP Tools, 36–37
- JSP wizard
 - writing pages, 211–213, 215
- JST (J2EE Standard Tools)
 - annotations, 91, 93
 - creating J2EE projects, 75
 - exporting projects, 83–84
 - importing projects, 82
 - J2EE Runtime Library, 79
 - JRE wizard, 75–77
 - New Project wizard, 80–81
 - project folder contents, 85–86
 - project properties, 84–85
 - Server Creation wizard, 77–79
 - working with projects, 87–88
- EJB Tools, 38–39
- flexible projects, 75
- goals, 27–28
- introduction, 25
- J2EE Module Tools, 29, 31
- J2EE Project Tools, 29, 31
- J2EE Server Tools, 31–32
- Java Web Services Tools, 39–40
 - Web Services Explorer, 40
- JCM (J2EE Core Model), 28
- JSP Tools, 36–37
- Navigation Tools, 41
 - J2EE Perspective, 41–42
 - Module View, 42
- project types, 74
- projects, 73
- relationship with WST, 48
- scope, 26
- Servlet tools, 32–36

- Web Services
 - consuming, 267
 - creating, 267
 - discovering, 267
- JSTL core library
 - configuring for web applications, 205–206
- JTA (Java Transaction API)
 - services provided, 8
 - version 1.0, 5
- K**
- Kunnumpurath, Meeraj and Spielman, Sue
 - Pro J2EE 1.4: From Professional to Expert, 162, 218
- L**
- <large-icon> tag, 185
- lib directory
 - installing JBoss application server, 314
- listeners.xml file, 203
- local home interface
 - generating with XDoclet, 104, 138–139
- local interface
 - generating with XDoclet, 103–104
- LocalObject interface
 - javax.ejb package, 90
- Lomboz, 19
- M**
- MapMessage interface, 158
- MDBs (message-driven beans), 155
 - overview, 160–162
 - writing, 163
 - finishing implementation, 172–174
 - MessageDrivenBean wizard, 163–170
 - XDoclet MDB annotations, 170–172
- Message interface, 158
 - javax.jms package, 161
- <message> tag, 280
- message-driven beans
 - generating, 167–169
 - introduction, 7
- message-driven beans. *See* MDBs
- <message-driven> tag, 185
- Message-Oriented Middleware (MOM), 155
- MessageConsumer class
 - registering listener, 159
- MessageDrivenBean interface
 - ejbRemove method, 162, 169
 - javax.ejb package, 160–161, 166, 169
 - setMessageDrivenContext method, 162, 169
- MessageDrivenBean wizard
 - writing MDBs, 163–170
- MessageDrivenContext interface
 - javax.ejb package, 162
- MessageListener interface
 - javax.jms package, 160, 166, 169
 - onMessage method, 161, 169
 - setMessageListener method, 160
- Middle Tier
 - J2EE architecture, 3
- mime-mappings.xml file, 203
- model-view-controller. *See* MVC
- <module> tag, 194
- Module View
 - access to navigation functionalities, 41
 - introduction, 42
- MOM (Message-Oriented Middleware), 155
- Mukhar, Kevin, Weaver, James L., and Crume, Jim
 - Beginning J2EE 1.4: From Novice to Professional, 162, 266
- MVC (model-view-controller)
 - introduction, 199
- MyEclipse, 19
- MySQL, 289
- N**
- NamingException class
 - javax.naming package, 97, 126
- Navigation Tools, 41
 - J2EE Perspective, 41–42
 - Module View, 42
- Navigator pane
 - Web Services Explorer, 287
- Navigator View
 - creating RDB perspective, 291
- NetworkServerControl class
 - org.apache.derby.drda package, 309
- New Connection wizard
 - sections, 292
 - user information section, 294
- New Java Class wizard
 - writing an action, 208–210
- New Project wizard
 - creating J2EE projects, 80–81
- nightly version
 - Eclipse build types, 65
- NoInitialContextException class
 - javax.naming package, 118
- O**
- OASIS (Organization for the Advancement of Structured Information Standard)
 - supported by WST, 47
- Object class
 - java.lang package, 169
- Object Selection Page
 - Web Services wizard, 275
- ObjectMessage interface, 158

onMessage method
 MessageListener interface, 161, 169
 openejb-jar.xml deployment descriptor, 182

OPTIONS
 supported by servlets, 221

Oracle, 289

org.apache.derby.drda package
 NetworkServerControl class, 309

org.apache.struts.action package
 Action class, 205
 ActionForm class, 205

Organization for the Advancement of Structured Information Standard.
See OASIS

Outline View
 creating RDB perspective, 291

P

packaging EJBs, 178–179
 exporting files, 180

packaging web applications. *See* web applications

persistence mechanism
 entity beans, 120

point-to-point message domain
 and JMS, 156

POJOs and JSP, 200

portability, benefits of J2EE Platform, 4

<portType> tag, 280

POST
 supported by servlets, 221
 using, 227

PostgreSQL, 289

primary keys
 entity bean must define, 120

Pro J2EE 1.4: From Professional to Expert
 Meeraj Kunnumpurath and Sue Spielman, 162, 218

project model
 WCM (Web Core Model), 47

.project property
 configurable properties for J2EE projects, 86

proxy configuration
 setting for WST Internet Tools, 53

publishing
 deploying web applications, 249
 EJBs, 186
 publish/subscribe model and JMS, 156

PUT, supported by servlets, 221

Q

Queue interface, 160

QueueConnectionFactory class, 160

R

RAR file
 creates a Connector Project, 83

RDB (relational databases), 289
 browsing a database, 295–298
 browsing a table, 298–300
 creating RDB perspective, 290–291
 data projects, 301–302
 executing statements, 302
 managing connections, 291–292, 294–295
 overview, 289–290
 SQL file types, 301

RDB Tools, 62
 Data Output View, 63–64
 Database Explorer View, 63
 SQL Scrapbook, 62

relational databases. *See* RDB

release version
 Eclipse build types, 65

remote interface
 generating with XDoclet, 102

remote session bean client, writing, 114
 call remote session bean for class, 116–118
 creating remote client project, 114–116

RemoteException class
 java.rmi package, 90

request handling
 and servlets, 221–225

Resource perspective
 creating RDB perspective, 291

responses, providing
 and servlets, 225–227

runtime environment
 deploying web applications, 241

.runtime property

configurable properties for J2EE projects, 86

S

SAAJ (SOAP with Attachments API for Java)
 version 1.2, 5

SCM (source configuration management)
 Team component, 14

<security-role> tag, 194

Server Creation wizard
 create a Tomcat server instance, 250
 creating J2EE Runtime Environment, 77–79
 creating server to host deployed EJBs, 187–190

server directory
 installing JBoss application server, 314

server model
 WCM (Web Core Model), 48

Server Tools, 50, 52

Service Deployment Configuration page
 Web Services wizard, 276

- service method
 - HttpServlet class, 227
- Service Oriented Architectures. *See* SOAs
- <service> tag, 280
- Servlet 2.4, 4
- Servlet interface
 - javax.servlet package, 219
- Servlet tools
 - introduction, 32–36
- Servlet wizard
 - writing servlets, 228–230, 232
- servlet-mappings.xml file, 203
 - creating in WEB-INF directory, 204
- ServletRequest interface
 - javax.servlet package, 218
- ServletResponse interface
 - javax.servlet package, 218
- servlets, 217
 - compared to JSPs, 217
 - introduction, 6
 - overview, 217–218
 - handling requests, 221–225
 - life cycle, 227
 - providing responses, 225–227
 - structure, 219–220
 - writing, 228
 - completing implementation, 233, 235, 238–239
 - Servlet wizard, 228–230, 232
 - XDoclet servlet annotations, 232–233
- servlets.xml file, 203
 - creating in WEB-INF directory, 204
- session beans, 89
 - converting to entity bean, 128–130
 - generating, 127
 - introduction, 7
 - overview, 89–90
 - writing, 93
 - EnterpriseJavaBean wizard, 93–104
 - finishing implementation, 113–114
 - remote session bean client, 114–118
 - XDoclet session bean annotations, 112–113
- Session class
 - createConsumer method, 159
 - createProducer method, 159
- Session Façade pattern, 89, 121
 - introduction, 90
- <session> tag, 185
- SessionBean interface
 - javax.ejb package, 127, 130
- setMessageDrivenContext method
 - MessageDrivenBean interface, 162, 169
- setMessageListener method
 - MessageListener interface, 160
- Simple Object Access Protocol (SOAP), 265
- Simple projects
 - support for, 74
- Skeleton Java Bean Service
 - creating, 268
- <small-icon> tag, 185
- SOAP (Simple Object Access Protocol)
 - utilized by Web Services, 265
- SOAP with Attachments API for Java (SAAJ), 5
- SOAs (Service Oriented Architectures)
 - servlets used in, 217
 - Web Services and, 264
- source configuration management tools
 - Team component, 14
- Spielman, Sue and Kunnumpurath, Meeraj
 - Pro J2EE 1.4: From Professional to Expert, 162, 218
- SQL (Structured Query Language), 289
 - file types, 301
- SQL Scrapbook, 62, 289
 - executing statements, 302
 - opening, 290
- SQL Server, 289
- SQL statement mapping
 - entity life cycle methods, 121
- SSE (Structured Source Editor)
 - JSP editor, 213
- stable version
 - Eclipse build types, 65
- Standard 1.1.x VM, 76
- Standard VM, 76
- Standard Widget Toolkit (SWT), 14
- stateful session beans, 89–90
- stateless session beans, 89–90
- Status pane
 - Web Services Explorer, 287
- StreamMessage interface, 158
- String class
 - java.lang package, 120
- Structured Query Language. *See* SQL
- Structured Source Editor Framework Tools, 59
 - features provided, 59
- Struts
 - creating an action class, 207–210
 - mapping action class, 210–211
 - overview, 201
 - web application configuration, 203–205
 - writing an action, 206
- Struts bean library
 - configuring for web applications, 205–206
- Struts HTML tag library
 - configuring for web applications, 205–206
- Struts JAR, 203
- Struts tag libraries, 201
- struts-config.xml file
 - basic file, 204
 - loaded by ActionServlet, 201

SWT (Standard Widget Toolkit)

Workbench layer, 14

Sysdeo, 19

T

taglibs.xml file, 203

<target> tag, 253

<task> tag, 35

Tasks View

creating RDB perspective, 291

TCP/IP monitoring

WST Internet Tools, 54–55

Team component

Eclipse Platform Runtime layer, 14

TextMessage interface, 158

textual data handled by PrintWriter object, 225

TheServerSide.com

Application Server Matrix, 20–21

thread management, 155

Tiles framework, 201

Topic interface, 160

TopicConnectionFactory interface

createTopicConnection method, 160

track-statements element, 319

Transfer Object pattern, 121

U

UDDI (Universal Description, Discovery, and Integration)

browsing registries with Web Services

Explorer, 285

utilized by Web Services, 265

utility class

generating with XDoclet, 104–108

V

validation of XML documents

Eclipse support for, 59

vendor independence

benefits of J2EE platform, 4

W

W3C (World Wide Web Consortium)

supported by WST, 47

WAR files

importing projects as Dynamic Web Project, 83

packaging web applications, 177, 241–242

WCM (Web Core Model), 47

Weaver, James L., Mukhar, Kevin, and Crume, Jim

Beginning J2EE 1.4: From Novice to Professional, 162, 266

web application configuration, 202

Struts, 203–205

tag libraries, 205–206

web application deployment descriptor, 202–203

web applications

accessing, 259–260

running with workbench, 260–261

deploying, 249–251

managing deployment with Ant, 251

build.xml, 252–254

integration with Eclipse, 254–256

targets, 254

working with Ant buildfiles, 258–259

packaging, 242–245

EAR files, 248–249

exporting artifacts, 249

web.xml, 245–247

web artifacts model

WCM (Web Core Model), 48

web browser

WST Internet Tools, 56

web components

containers, 8

introduction, 6

web container

deploying web applications, 241

Web Core Model (WCM), 47

web event listeners

introduction, 6

web modules

deployment modules, 9

Web Projects

types supported by WST, 49–50

Web Service Client page

Web Service Client wizard, 284

Web Service Client Test page

Web Service Client wizard, 284

Web Service Client wizard

consuming Web Services, 281–284

Web Service Description Language (WSDL), 265

Web Service Java Bean Identity page

Web Services wizard, 277

Web Service Proxy Page

Web Service Client wizard, 283

Web Service Selection Page

Web Service Client wizard, 282

Web Service Tools, 40, 61–62

Web Services, 263

consuming, 281

Web Service Client wizard,

281–284

creating, 267–268

bottom-up approach, 267

EJBs expose methods as, 99

overview, 264–266

support in JST, 266

Web Services wizard, 268–280

- Web Services Explorer
 - introduction, 285–288
 - Java Web Services Tools, 40
 - Web Services for J2EE
 - version 1.1, 5
 - Web Services Interoperability Organization (WS-I), 47
 - Web Services page
 - Web Services wizard, 275
 - Web Services wizard
 - creating Web Services, 268–280
 - Web Standard Tools. *See* WST
 - Web Tier
 - J2EE architecture, 3
 - Web Tools, 57–58
 - Web Tools Platform. *See* WTP
 - WEB-INF directory
 - location for deployment descriptor, 243
 - used to hold third-party libraries, 244
 - web-security.xml file, 203
 - web-settings.xml file, 203
 - web.xml file, 202
 - configuring ActionServlet, 203
 - contents of, 245
 - important to smooth functioning of web application, 245
 - weblogic-cmp-jar.xml deployment descriptor, 182
 - weblogic-ejb-jar.xml deployment descriptor, 182
 - WebSphere Studio Application Developer (WSAD), 20
 - welcomefiles.xml file, 203
 - Workbench layer
 - introduction, 13–14
 - Workspace layer
 - introduction, 13
 - World Wide Web Consortium (W3C), 47
 - WS-I (Web Services Interoperability Organization)
 - supported by WST, 47
 - WSAD (WebSphere Studio Application Developer), 20
 - WSDL (Web Service Description Language)
 - utilized by Web Services, 265
 - WSDL Creation wizard
 - creating a Web Service, 268
 - WSDL editor, 280
 - WST (Web Standard Tools)
 - creating and editing HTML documents, 221
 - goals, 46
 - Internet Tools, 52
 - Cache Resolver, 55
 - proxy settings, 52–53
 - TCP/IP monitoring, 54–55
 - web browser, 56
 - introduction, 45
 - RDB Tools, 62
 - Data Output View, 63–64
 - Database Explorer View, 63
 - SQL Scrapbook, 62
 - relationship with JST, 48
 - Server Tools, 50, 52
 - Simple projects, 74
 - Structured Source Editor Framework Tools, 59–61
 - technologies supported, 46
 - WCM (Web Core Model), 47
 - Web Projects, 49–50
 - Web Service Tools, 61–62
 - Web Tools, 57–58
 - XML Tools, 58
 - WTP (Web Tools Platform)
 - contributing, 23
 - creating extension, 67–69
 - dependencies, 65
 - goals, 20
 - JST scope, 22–23
 - WST scope, 21–22
 - history, 19–20
 - installing, 70
 - installing dependencies
 - EMF/SDO/XSD, 69
 - GEE, 70
 - JEM, 70
 - installing JBoss, 71–72
 - installing XDoclet, 72
 - support for XDoclet annotations, 94, 123
 - .wtpmodules property
 - configurable properties for J2EE projects, 86
- X**
- XDoclet, 23
 - annotations, 98–99, 123
 - configuring location and version, 92
 - container-specific notations, 130
 - entity bean annotations, 151–152
 - generates interfaces, 102
 - home interface, 102
 - local home interface, 104
 - local interface, 103–104
 - remote interface, 102
 - generating deployment descriptors, 182
 - generating DTO (Data Transfer Object), 140–143
 - generating ejb-jar.xml file, 108–110, 146–149
 - generating jboss.xml File, 110–111
 - generating local home interface, 138–139
 - generating utility class, 104–108, 143–146
 - identifying the primary key field, 132

- installing, 72
 - MDB annotations, 170–172
 - merge files, 203
 - servlet annotations, 232–233
 - session bean annotations, 112–113
 - supported by JST, 27
- XDoclet Builder, 91
- adding Struts ActionServlet in a JST Web Project, 204
- XML Schema Definitions. *See* XSD
- XML Tools, 58
- XPath
- Eclipse support for, 58
- XQuery
- Eclipse support for, 58
- XSD (XML Schema Definitions)
- Eclipse support for, 58
- XSLT (Extensible Stylesheet Language Transformation)
- Eclipse support for, 59