The DataSource Interface

The DataSource interface, introduced in JDBC 2.0 Optional Package, is the preferred approach to obtaining data source connections. A JDBC driver that implements the Datasource interface returns connections that implement the same interface, Connection, as those returned by a DriverManager using the Driver interface. Using a Datasource object increases application portability by making it possible for an application to use a logical name for a data source instead

of having to supply information specific to a particular driver. A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory InterfaceTM (JNDI). The DataSource object, represents a physical data source and provides connections to that data source. If the data source or information about it changes, the properties of the DataSource object can simply be

modified to reflect the changes; no change in application code is necessary.

The DataSource interface can be implemented so that it transparently provides the following:

■ Increased performance and scalability through connection pooling

■ Support for distributed transactions through the XADataSource interface

The next three sections discuss (1) basic DataSource properties, (2) how logical naming using the JNDI API improves an applications portability and makes it easier to maintain, and (3) how to obtain a connection.

DataSource Properties

The JDBC API defines a set of properties to identify and describe a DataSource implementation. The actual set required for a specific implementation depends on the type of DataSource object, that is, whether it is a basic DataSource object, a ConnectionPoolDataSource object, or an XADataSource object. The only property required for all DataSource implementations is description.

The following table describes the standard DataSource properties:

**Property Name Type Description**

databaseName String name of a particular database on a server

dataSourceName String a data source name; used to name an underlying

XADataSource object or ConnectionPoolDataSource object when pooling of connections is done description String description of this data source networkProtocol String network protocol used to communicate with the server password String a database password

portNumber int port number where a server is listening for requests

roleName String the initial SQL rolename serverName String database server name user String user’s account name

DataSource properties follow the convention specified for properties of JavaBeans convention. DataSource implementations may augment this set with implementation-specific properties. If

new properties are added, they must be given names that do not conflict with the standard property names.

DataSource implementations must provide “getter” and “setter” methods for each property they support. These properties typically are initialized when the DataSource object is deployed, as in CODE EXAMPLE 9-5, in which a VendorDataSource object implements the DataSource interface.

VendorDataSource vds = new VendorDataSource();

vds.setServerName("my\_database\_server");

String name = vds.getServerName();

Setting and getting a DataSource property

DataSource properties are not intended to be directly accessible by JDBC clients.

This design is reinforced by defining the access methods on the implementation class rather than on the public DataSource interface used by applications.

Furthermore, the object that the client manipulates can be a wrapper that only implements the DataSource interface. The setter and getter methods for the properties need not be exposed to the client.

Management tools that need to manipulate the properties of a DataSource implementation can access those properties using introspection.

The JNDI API and Application Portability

The Java Naming and Directory Interface (JNDI) API provides a uniform way for applications to access remote services over the network. This section describes how it is used to register and access a JDBC DataSource object.

Using the JNDI API, applications can access a DataSource object by specifying its logical name. A naming service using the JNDI API maps this logical name to a corresponding data source. This scheme greatly enhances portability because any of the DataSource properties, such as portNumber or serverName, can be changed without impacting the JDBC client code. In fact, the application can be re-directed to a different underlying data source in a completely transparent fashion. This is particularly useful in the three-tier environment, where an application server hides the details of accessing different data sources.

Following Example illustrates the use of a JNDI-based naming service to deploy a new VendorDataSource object.

// Create a VendorDataSource object and set some properties

VendorDataSource vds = new VendorDataSource();

vds.setServerName("my\_database\_server");

vds.setDatabaseName("my\_database");

vds.setDescription("data source for inventory and personnel");

// Use the JNDI API to register the new VendorDataSource object.

// Reference the root JNDI naming context and then bind the

// logical name "jdbc/AcmeDB" to the new VendorDataSource object.

Context ctx = new InitialContext();

ctx.bind("jdbc/AcmeDB", vds);

Registering a DataSource object with a JNDI-based naming service

**Note –** Java EE components use a special convention for naming their data sources

Getting a Connection with a DataSource Object

Once a DataSource object has been registered with a JNDI-based naming service, an application can use it to obtain a connection to the physical data source that it represents, as is done in the following example.

// Get the initial JNDI naming context

Context ctx = new InitialContext();

// Get the DataSource object associated with the logical name

// "jdbc/AcmeDB" and use it to obtain a database connection DataSource ds = (DataSource)ctx.lookup("jdbc/AcmeDB");

Connection con = ds.getConnection("user", "pwd");

Getting a Connection object using a DataSource object

The DataSource implementation bound to the name “jdbc/AcmeDB” can be modified or replaced without affecting the application code.