RDM Server 8.3

JDBC Guide

Trademarks

Contents

Introduction to Java Database Connectivity with RDM Server

Java Database Connectivity™ provides a programming level interface for database access that is uniform and standard. Like ODBC, JDBC is based on the X/Open SQL Call level interface. More information on JDBC is available from the JavaSoft web site (http://www.javasoft.com/).

RDM Server supports a "Native-Protocol All-Java Driver" (type 4 JDBC driver). The type 4 JDBC driver is completely implemented in Java code and communicates directly with the database using RDM Server's network protocol.

Creating a Database Application Using the RDM Server JDBC Driver

A Java applet is a program written in Java that is part of an HTML page residing on a Web server. Users run the applet by browsing the applet's Web page. When a user downloads the Web page, the browser also downloads the applet and its JDBC Driver, and then executes the applet within the browser context. Since the applets usually come across the network from unidentifiable servers, they are sandboxed on the client to provide security. This means that there are numerous constraints when database applications are deployed as applets.

Unlike an applet, a Java application is a program that resides on the user's machine. Thus it does not have the limitations of an applet. However, since Java is an interpreted language, users must start Java applications from a command line. To do this, pass the name of the application as a parameter when starting the Java interpreter.

To develop an RDM Server database applet or application, you must use a Java development environment compatible with Version 5 or later of JavaSoft's Java Development Kit (JDK).

To create a database applet or application program to access RDM Server, the following steps provide a general guideline.

1. Register the JDBC Driver.
2. Use the RDM Server JDBC URL syntax.
3. Connect to the server.
4. Create a statement handle.
5. Prepare and execute any SQL statements.
6. Get data results from the SQL execution.

Important: Please refer to the JDBC specification before reading further in this document. This document assumes basic knowledge of JDBC interfaces and classes. A complete JDBC API reference is available in the JDK or at the JavaSoft website (http://www.javasoft.com).

Registering the JDBC Driver

If you are using Java 6 or later just having the driver file in your CLASSPATH is sufficient to register it. This is because starting in Java 6, JDBC was extended to support the Service Provider mechanism and the Birdstep JDBC driver contains sufficient information for Java to know that it is a JDBC driver and register it automatically. If you are using a previous version of Java, you will have to ensure that the JDBC driver gets registered. To register the JDBC Driver with the JDBC Driver Manager, you simply ensure that the JDBC Driver gets loaded. The simplest way to do this is with the **forName()** method of the **Class** class. If you are unsure whether you will be using Java 6 or not you can use this method. It will not conflict with the Service Provider mechanism.

The following code example shows how to register the JDBC Driver.

Example 1

In this example, you register the JDBC Driver before calling the **DriverManager.getConnection()** method.

```
Connection login(String sUser, String sPassword)
    throws SQLException, ClassNotFoundException
{
    String sURL = "jdbc:birdstep://localhost:1530";

    Class.forName("com.birdstep.rdms.jdbc.Driver");
    return DriverManager.getConnection(sURL, sUser, sPassword);
}
```

RDM Server JDBC URL Syntax

The Uniform Resource Locator (URL) you pass to the **DriverManager.getConnection()** method must have the syntax shown below. This syntax is the same for all three overloads (versions) of the method. Refer to Example 2.

(In the next section, Connecting to the Server, a complete description is given of the overloads and data connection properties you can specify.)

Syntax:

```
jdbc:birdstep://host-list[?property-list]
```

where:

host-list

A comma-separated list of RDM Server(s) with the following format: host1[:port1][,host2[:port2],…hostn[:portn]]

hostx

The DNS name, host name, or IP address of a JDBC Listener Process machine.

portx

An optional port number of the JDBC Listener Process on that machine. If no port is specified, the default port value (1530) is used.

server

The name or alias of the RDM Server database server you want to connect to.

property-list

An ampersand-separated list of property/value pairs with the following format: prop1=val1[[&prop2=val2]&…propn=valn] See RDM Server Data Connection Properties later in this document for descriptions of the properties you can specify

propx

The name of one of the RDM Server data connection properties.

valx

A valid value for the property.

1) In the URL, you can add the **birdstep.** prefix to any property name except the user and password. 2) For more information on JDBC URL syntax refer to the JavaSoft web site (http://www.javasoft.com/).

Example 2

```
Connection login(String user, String password) throws SQLException
{
    String url = "jdbc:birdstep://localhost:1530?user=" + user +
            "&password=" + password + "&birdstep.autocommit=true";

    try {
        Class.forName("com.birdstep.rdms.jdbc.Driver");
    }
    catch (ClassNotFoundException e) {
        System.out.println("Birdstep JDBC driver not loaded");
        return null;
    }
    return DriverManager.getConnection(url);
}
```

Connecting to the Server

Connect to your server by passing the following information to one of the overloads of the **DriverManager.getConnection()** method. If **getConnection()** is successful, it will return an object instance of a **Connection** class; this instance can be used to create multiple instances of the **Statement** class that implemented the **Statement** interface.

Table 1-1. Server Connection Information

Information	Passed Where?	Required?
The hostname and port number of the RDM Server (or a list of servers for fault tolerance)	Only in the URL	Required
A user ID and password recognized by the server	As explicit parameters for overload 1 or as properties: In the URL or In a Properties object	Required
Additional property values (including the name of a JDBC data profile)	In the URL or In a Properties object	Optional

Creating a Statement Handle

Once you connect to the RDM Server database server successfully, the bulk of your database programming involves these tasks: executing SQL statements, processing data resulting from SQL query executions, and error handling.

To process SQL statements, you must create a **Statement** class.

Example 3 creates an instance of a class that implements the **PreparedStatement** interface and uses this class to prepare an SQL statement and execute it.

Example 3

```
void prepareAndExecute(Connection conn)
{
    PreparedStatement pstmt;
    InputStream       istream;
    String            file = "Test_rdms_jdbc.java";

    try {
        istream = new FileInputStream(ifile);
    }
    catch (FileNotFoundException e) {
        System.out.println("The file '" + ifile + "' does not exist");
        return;
    }

    try {
        // Prepare SQL statement: if successful a PrepareStatement is
        // returned
        pstmt = conn.prepareStatement("insert into mytable " +
                "(timecol, datecol, chartext) values (?, ?, ?)");

        //Bind values
        java.util.Date now = new java.util.Date();
        pstmt.setTime(1, new java.sql.Time(now.getTime()));
        pstmt.setDate(2, new java.sql.Date(now.getTime()));
        pstmt.setAsciiStream(3, istream);

        // execute the SQL statement
        pstmt.execute();

        // if autocommit was off you could do a "conn.commit();" here to
        // cause the data to be committed.

        pstmt.close();
```

```
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Getting Results of an SQL Execution

Resulting data (if any) from an SQL query is returned to the JDBC application via a ResultSet. Example 4 executes an SQL query that returns data to the JDBC application.

Example 4

```
public void testReadBlobs(Connection conn)
{
    InputStream instream;

    try {
        //Create a Statement from the Connection
        Statement stmt = conn.createStatement();

        //Execute the query and return the resulting data into ResultSet
        ResultSet results = stmt.executeQuery(
                "select timecol, datecol, chartext from mytable");

        while (results.next()) {
            System.out.println(results.getString("timecol"));
            System.out.println(results.getString("datecol"));
            System.out.println("ASCII Data:");
            instream = results.getAsciiStream("chartext");
            BufferedReader br = new BufferedReader(
                    new InputStreamReader(instream));

            String line;
            try {
                while ((line = br.readLine()) != null)
                    System.out.println("  " + line);
            }
            catch (IOException e) {
                System.out.println("I/O error reading the chartext column");
                stmt.close();
                return;
            }
        }

        stmt.close();
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
}
```

RDM Server Data Connection Properties

This section describes the data connection properties you can set in your Java applet or application. Each of the property descriptions contains the name of the property and what it sets, possible values, default value (if any), and additional comments on usage.

user		Sets the AuthID with which the application attempts to login to the RDM Server.
	Possible Values	Any valid AuthID with the appropriate access to the data referenced in your application.
	Default Value	None
	Comments	You must specify a value for this property.

password		Sets the password for the AuthID specified by the value of the user property.
	Possible Values	The current password for the AuthID specified as the value of the user property. For an AuthID that has no password, use an empty string as the value of this property.
	Default Value	None
	Comments	You must specify a value for this property.

autocommit		Sets the autocommit mode.
	Possible Values	ON, TRUE, OFF, FALSE
	Default Value	TRUE
	Comments	This property and its default value match the JDBC standard. Auto-commit causes the JDBC driver to do a transaction commit after each individual SQL statement.

transIsolation		Sets the transaction isolation level.
	Possible Values	REPEATABLE_READ, READ_COMMITTED, or READ_UNCOMMITTED
	Default Value	REPEATABLE_READ
	Comments	This property and its default value match the JDBC standard. Transaction isolation levels define how isolated each transaction is from the data accessed and modified by other transactions.

Overloads of the getConnection Method

In a Java applet or application, you connect to your server by calling one of three overloads for the **DriverManager.getConnection** method.

The one you choose to use depends upon the following:

- The parameters supported by your Java development environment, and
- The data connection properties you need to specify.

The overloads of the **DriverManager.getConnection()** method take the following parameters:

1. A URL string, a user ID string, and a password string
2. A URL string and a Properties object
3. A URL string only

Overload 1: URL, User ID, and Password

You can use Overload 1 to specify all of the following without having to define a Properties object:

- The URL (string) that identifies the address of the RDM Server and port number.
- The user ID with which to log into the server (string).
- The password associated with the user ID (string).
- Other RDM Server data connection properties defined in the URL.

It is appropriate to use Overload 1 if you:

Have few property values to set, or

Need to switch from the JDBC Driver to other JDBC Drivers within the code of your applet or application.

Example 5

Example 5 uses Overload 1 to define property values in the URL, that is, without instantiating a Properties object.

```
Connection login(String user, String password)
{
    String url = "jdbc:birdstep://myhost:1530";

    //Connect to the database
    return DriverManager.getConnection(url, user, password);
}
```

Overload 2: URL and Properties Object

You can use Overload 2 to specify:

- The server URL (string), or
- The RDM Server data connection properties defined in a single instance of a Properties object.

It is appropriate to use Overload 2 if:

Your Java development environment supports a Properties object and you have many properties you want to set for each data connection.

In addition to specifying the property values in an object, you can:

- Override property values in the URL itself, or
- Specify the name of a JDBC data profile in your Properties object to pick up predefined property values.

Example 6

Example 6 uses Overload 2 to set properties in an object and pass them along with the URL.

```
Connection login(String user, String password)
{
    Properties props;
    String url = "jdbc:raima://myhost:1530";

    //Create a Properties object
    props = new Properties();

    //Load properties from a properties file
    props.load(new FileInputStream("mydb.properties"));

    //Set the user name and password
    props.put("user", user);
    props.put("password", password);

    //Connect to the database
    return DriverManager.getConnection(url, props);
}
```

Important: In the Properties object you must add the **birdstep.** prefix to every property name except user and password.

Overload 3: URL Only

If necessary, you can specify the AuthID and password for logging into the server (as well as any other data connection properties), in the URL itself. As with the other overloads, one of the properties you can include in the URL is the name of a JDBC data profile.

Example 7

Example 7 uses Overload 3 to define property values and the user name and password in the URL, that is, without instantiating a Properties object.

```
Connection login()
{
    String url = "jdbc:birdstep://myhost:1530?user=admin&password=secret";

    //Connect to the database
    return DriverManager.getConnection(url);
}
```