THE POWER TO KNOW.

SAS® 9.3 Drivers for JDBC
Cookbook

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 Drivers for JDBC: Cookbook* . Cary, NC: SAS Institute Inc.

SAS® 9.3 Drivers for JDBC: Cookbook

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, July 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents

What's New for the SAS 9.3 Drivers for JDBC

Overview

The SAS 9.3 release of the SAS Drivers for JDBC includes updated Java version support, a feature to enhance the fetch size for the IOM driver, and documentation enhancements.

Supported Java Versions

SAS 9.3 supports Java 6 as the baseline Java version. The SAS Drivers for JDBC work with Java 6 and are backward compatible with Java 5.

Enhanced Fetch Size for the IOM Driver

When using the IOM driver (com.sas.rio.MVADriver), the default behavior for the driver is to calculate an optimal fetch size at run time. The driver calculates the fetch size by setting the fetch size to the number of rows that fit in a 16-kilobyte buffer. This behavior can be overridden with the setFetchSize method.

Documentation Enhancements

Section "Using Timestamps, Dates, and Times" on page 30 provides information about SAS dates and times. Sample code for creating tables with time-related columns and inserting values is provided.

Chapter 1

Introduction to the SAS Drivers for JDBC

About the SAS Drivers for JDBC: Cookbook

How the Cookbook Can Help You Write Applications

The *SAS Drivers for JDBC: Cookbook* provides programming recipes for use with the SAS Drivers for JDBC. The recipes are designed to expedite development efforts in these ways:

- by providing ready-to-use sample code, as complete programs or as fragments
- by encouraging best practices
- by answering frequently asked questions

You can use the recipes directly or modify them to fit your needs.

What You Should Know in Order to Use This Cookbook

The cookbook assumes that the following statements are true:

- You are familiar with the JDK (Java Development Kit).
- You know how to compile and run Java applications.
- You are familiar with Oracle JDBC API documentation, such as the information available from **`http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/`**.
- Either you know how to start the SAS server that you want to access or the server has been started for you.

About the SAS Drivers for JDBC

The SAS drivers for JDBC implement data access interfaces that conform to the JDBC API (application programming interface) from Oracle.

- The SAS/SHARE driver for JDBC supports multi-user, forward-only access to Base SAS and Scalable Performance Data (SPD) Server data via the SAS/SHARE server.
- The SAS Integration Technologies IOM driver for JDBC supports multi-user, scrollable access to Base SAS data via a SAS Workspace Server. If you have licensed one or more SAS/ACCESS engines, you can also use the IOM driver to access third-party relational data.

The SAS drivers for JDBC are native-protocol, fully Java technology-enabled drivers. They directly convert JDBC technology calls into the network protocol that is used by the underlying data sources. The conversion enables client machines to make direct calls to the data servers.

About the Data That Can Be Accessed by Using the SAS Drivers for JDBC

Base SAS Data Sets

A Base SAS data set is the proprietary file format for SAS software. Data sets contain data values that are organized as a table of observations (rows) and variables (columns). The supported file format is the same as SAS data sets created by the BASE engine Version 7 and higher. A supported SAS data set uses the extension **`.sas7bdat`**.

SAS Scalable Performance Data Server Data

SPD Server data provides performance for very large amounts of data. The data source provides threaded I/O processing, using multiple CPUs in order to read data and deliver it rapidly to applications. Very large files can be processed, because the data can span volumes but is referenced as a single file. The data in the files is also partitioned, which allows the data to be read in multiple threads per CPU.

Third-Party Relational Data

Third-party relational database management systems (DBMS) data that can be accessed includes DB2 and Oracle.

Accessibility Features in the SAS Drivers for JDBC

The SAS Drivers for JDBC include accessibility and compatibility features that improve usability of the product for users with disabilities. These features are related to

accessibility standards for electronic information technology adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973, as amended. If you have questions or concerns about the accessibility of SAS products, send e-mail to accessibility@sas.com.

Chapter 2

System Requirements and Installation

System Requirements

This documentation assumes that you have an existing SAS environment that includes a local or remote SAS server for accessing data. The server determines which driver you use in your application as explained in the following table.

Table 2.1 *SAS Servers and Their Associated Drivers*

SAS Server	SAS Driver for JDBC
SAS Workspace Server	IOM Driver
SAS/SHARE server	SAS/SHARE Driver

For information about how to start the SAS servers, see the following resources:

Table 2.2 *Documentation Resources for Starting SAS Servers*

Server	Documentation Title	Section
SAS Workspace Server	*SAS Intelligence Platform: System Administration Guide*	"Using SAS Management Console to Operate SAS Servers" (refer to the information about the SAS Object Spawner because it creates the SAS Workspace Server process)
SAS/SHARE Server	*SAS/SHARE: User's Guide*	"Starting a Server: A Fast-Track Approach"

These documents are available from SAS Web site at **`http://support.sas.com/documentation/index.html`**.

Here are some additional system requirements:

- To access data through the SAS Workspace Server, a SAS Workspace Server must be licensed and installed.
- To access data through a SAS/SHARE server, a SAS/SHARE server must be licensed and installed.
- JRE 1.5 or later must be installed.
- A CLASSPATH environment variable must be properly configured.

Location of the Installation Packages

Download the most current versions of the drivers from **`www.sas.com/apps/demosdownloads/setupintro.jsp`**. Click **SAS Drivers for JDBC**.

Setting the CLASSPATH Environment Variable

The CLASSPATH is the path that the Java Runtime Environment (JRE) searches in order to find CLASS, JAR, and other resource files. You can use one of these two methods to set the CLASSPATH:

- Use the **`-classpath`** option with a software development kit (SDK) tool such as **`java`**. This method enables you to set the CLASSPATH for each individual application.
- Set the CLASSPATH environment variable. This recommended method makes the CLASSPATH available for all applications by default.

When setting the CLASSPATH as an environment variable, follow these guidelines:

- Set the CLASSPATH as an environment variable for the user (not the system).
- Do not forget to precede the CLASSPATH with a period (.). If you leave out the period and you invoke Java from the same location as your application code, you get the error: `java.lang.NoClassDefFoundError`. This requirement applies only when the JAR file is in the current directory.
- Add the SAS Drivers for JDBC JAR files to the CLASSPATH. You cannot point to the directory that contains the JAR files.
  - The SAS/SHARE driver needs sas.core.jar and sas.intrnet.javatools.jar.
  - The IOM driver needs sas.core.jar and sas.svc.connection.jar. If the Security Support Provider Interface (SSPI) feature is used, then sas.security.sspi.jar is also needed.

Here are some considerations:

- If there is a conflict, the order in which the JAR files appear on the CLASSPATH determines which class is used.
- If you change the CLASSPATH, you do not need to reboot. However, if you are running from a command prompt, you must bring up a new command prompt. If you are running an applet, you might need to close your Web browser and then restart it.

Chapter 3

Learning about SAS Connections

What You Need to Know About SAS Connections

In most cases, the SAS Drivers for JDBC can be used to perform tasks as described in the JDBC specification, with the only difference being the connection information. You use the connection properties in order to control the behavior of a SAS driver for JDBC.

After the connection is made, most of the JDBC code will be the same from driver to driver (assuming that the servers support the desired functionality). This chapter contains the following connection information that is specific to the SAS drivers.

- the class names for each driver
- a list of connection properties for each driver
- how to use the JDBC URL when making connections, including sample code

JDBC Driver Class Names

JDBC driver class names are used to load the appropriate driver into the Java session. Class names are unique to the driver. The following table lists the JDBC driver class names for each SAS driver.

SAS Driver	Driver Class Name
IOM driver	com.sas.rio.MVADriver
SAS/SHARE driver	com.sas.net.sharenet.ShareNetDriver

Connection Properties for the SAS Drivers

IOM Driver Properties

The following table describes the driver connection properties that are available for the IOM driver.

Table 3.1 *Connection Properties for the IOM Driver (in Alphabetical Order)*

Property	Description	Required
applyFormats	Specifies whether column formats are applied when retrieving Base SAS data. If the applyFormats connection property is set to **`true`**, then formatted data is returned from the SAS Workspace Server. It is returned as a String with column formatting (as defined in the data set) applied. By default, the applyFormats connection property is set to **`false`**, and formats are not applied when data is retrieved.	No
dbms	Specifies the name of the engine that is used to connect directly to the underlying third-party relational DBMS such as ORACLE or MYSQL. The value must be eight characters padded with trailing blanks. If you do not set this value, then the default value of **`sqlview`** is used and your Java application can access only Base SAS data sets.	No
dbmsOptions	Specifies valid options for the engine that is specified by using the **`dbms`** connection property. Specify the options as you would specify the database-definition arguments when using the PROC SQL CONNECT TO statement. See the “Pass-Through Facility for Relational Databases” in *SAS/ACCESS for Relational Databases: Reference*.	No

Property	Description	Required
encryptionAlgorithms*	Specifies a comma-separated list of encryption algorithms in order of preference. Your Java application will use this list when negotiating encryption algorithms with the server. List only the algorithms that you want to use for a particular connection. If no algorithms are listed, then the server will choose one. Here are the possible values for this property: sasproprietary a stream cipher developed at SAS rc2 a block cipher developed by RSA Data Security rc4 a stream cipher developed by RSA Data Security des a block cipher known as Data Encryption Standard tripledes DES applied three times with separate keys aes a block cipher that encrypts data in blocks of 128 bits by using a 256-bit key	No
encryptionContent	Specifies which messages should be encrypted if encryption is used. Here are the valid values: all encrypt all messages. This value is the default. authentication encrypt only messages that contain user name and password information.	Yes, if encryption is used
encryptionPolicy	Specifies whether your Java application will attempt to negotiate and use an encryption algorithm with the server and what to do if the negotiations are not successful. Here are the possible situations: • If the value of this property is **`none`**, then your Java application will not attempt to negotiate and use an encryption algorithm with the server. If the server requires encryption to be used, then the connection will fail. This value is the default. • If the value of this property is **`optional`**, then your Java application will attempt to negotiate and use an encryption algorithm with the server. If the negotiations fail, then your Java application will try to continue with an unencrypted connection. However, the unencrypted connection will also fail if the server requires encryption. • If the value of this property is **`required`**, then your Java application will attempt to negotiate and use an encryption algorithm with the server. If the negotiations fail, then the connection fails.	No

Property	Description	Required
librefs	Specifies the name and location of one or more SAS libraries to assign.	Yes, if the server does not already have the librefs defined and you are not accessing data in the Work libref
password	Specifies the password that corresponds to the user name that you specify by using the user connection property.	Yes, if the server is secure
remarks	Specifies a Boolean value that determines whether the IOM driver returns the Base SAS data set label for each data set in the library that your application accesses. If the value is **`true`**, then the labels are returned. When this property is set to **`true`**, operations that require reading and returning data set labels can result in poor performance. This is the case because each data set must be opened in order to read the label. For example, if this property is set to **`true`**, performance might be degraded when using the MVADatabaseMetaData.getTables() method to retrieve a description of the tables in a catalog. The default value of this property is **`false`**.	No
spn	Set the service principal name to use when connecting to this server (only applicable for SSPI when using Kerberos or Negotiate).	No
undoPolicyNone	Specifies a Boolean value that determines whether changes to the data are undone when an UPDATE or INSERT statement generates errors. When the value is set to **`true`** (the default value), the changes are not undone. When the value is set to **`false`**, the server attempts to undo the changes.	No
user	Specifies a valid user name for the SAS Workspace Server that you are connecting to.	Yes, if the server is secure
usesspi	Tells the driver whether to use Security Support Provider Interface (SSPI) or not for authentication: none Do not use SSPI. This is the default value. Kerberos Use Kerberos authentication. NTLM Use Microsoft NT LAN Manager authentication. negotiate Enables the client and server to negotiate a site-specific package to use.	No

* To use RC2, RC4, DES, TRIPLEDES, or AES, you must license SAS/SECURE on the server and install the Java component of SAS/SECURE (sasecjav.zip) in your code base. The availability of SAS/SECURE is subject to import and export restrictions.

SAS/SHARE Driver Properties

The following table describes the driver connection properties that are available for the SAS/SHARE driver.

Table 3.2 *Connection Properties for the SAS/SHARE Driver (in Alphabetical Order)*

Property	Description	Required
appname	Specifies the name of the Java applet or application that you are creating. This name will appear in all server log entries that are related to the applet or application. The value can be a maximum of eight characters. If the value is longer than eight characters, then the name will be truncated.	Not required, but provides information for the SAS log
dbms	Specifies the engine. For example, you can specify **spds**, which is the name of the engine that is used to connect directly to the SPD Server. The default is **sqlview**, which specifies that SAS data sets will be accessed.	No
dbmsOptions	Specifies valid options for the engine that is specified by using the **dbms** connection property. Specify the options as you would specify the database-definition arguments when using the PROC SQL CONNECT TO statement. See the "Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.	No
librefs	Specifies the name and location of one or more SAS Libraries to assign.	Yes, if the server does not already have the librefs defined and you are not accessing data in the Work libref
password	Specifies the password that corresponds to the user name that you specify by using the user connection property. Typically thought of as the password that is required to connect to the server, this password is required by the network protocol.	Yes, if the server is secure
pt2dbpw	Specifies the password that is required by the third-party relational database management system (DBMS) that is specified by the **dbms** connection property. This password is the SQL Pass-Through Facility password for the SAS/SHARE server that your Java application will connect to. The SQL Pass-Through Facility enables your application to access third-party DBMSs. See the "Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.	Yes, if the DBMS connection requires it

Property	Description	Required
remarks	Specifies a Boolean value that determines whether the SAS/SHARE driver for JDBC returns the Base SAS data set label for each data set in the library that your application accesses. If the value is **`true`**, then the labels are returned. When this property is set to **`true`**, operations that require reading and returning data set labels can result in poor performance. This is the case because each data set must be opened in order to read the label. For example, if this property is set to **`true`**, performance might be degraded when using the MVADatabaseMetaData.getTables() method to retrieve a description of the tables in a catalog. The default value of this property is **`false`**.	No
routerUrl*	Specifies the URL of the Message Router. Use this form: `(routerUrl http://your_server/cgi-bin/shrcgi)` where *http://your_server/cgi-bin/shrcgi* is the URL of the Message Router (shrcgi.exe), one of the tunnel feature's server programs.	No
sapw	Specifies the Server Access Password that is required if the SAS/SHARE server was started with the User Access Password (UAPW) option. More often, the password connection property will be used instead.	Yes, if the server is secure
undoPolicyNone	Specifies a Boolean value that determines whether changes to the data are undone when an UPDATE or INSERT statement generates errors. When the value is set to **`true`** (the recommended setting), the changes are not undone. When the value is set to **`false`**, the server attempts to undo the changes.	No
user	Specifies a valid user name for the SAS/SHARE server that you are connecting to.	Yes, if the server is secure

* The tunnel feature consists of programs that are installed on your Web server. These programs use the Common Gateway Interface (CGI) to receive requests from the Java applet that is running on a user's browser. The server programs forward the requests to the SAS server for processing. When the processing is complete, the programs return the results to the applet.

Passing Connection Properties on the JDBC URL

What Is the JDBC URL?

The JDBC URL is a method of supplying connection information to the data source that you are accessing. It is an alternative to providing a connection property list. The JDBC URL uses this form:

jdbc:subprotocol:subname

CAUTION:

Do not pass properties through the URL if you are creating a connection Properties object. Properties that are passed through the URL override properties that are specified in the Properties object used in the DriverManager.getConnection(String url, Properties info) method. For example, if you use both methods of specifying properties, a libref that is specified in a URL overrides any libref that is also specified in the connection Properties object.

What Is the Subprotocol?

The *subprotocol* is unique to the server. The subprotocols for the SAS servers are listed in the following table.

Table 3.3 *Subprotocols*

SAS Server	Subprotocol
SAS Workspace Server	sasiom
SAS/SHARE server	sharenet

What Is the Subname?

The *subname* uses the form *//hostname:port*. The *host name* and the *port* are unique to your site and should be known by your database administrator. For example, the following JDBC URL might be used for a SAS Workspace Server with the host name **`c123.na.abc.com`** using port number **`5671`**.

```
jdbc:sasiom://c123.na.abc.com:5671
```

The JDBC URL Syntax

Here is the syntax for passing connection properties on the JDBC URL:

```
"url?[property=value]&[...]"
```

Here is an example that includes user and password properties:

```
"jdbc:sasiom://c123.na.abc.com:5671?user=jdoe&password=4ht8d"
```

Sample Code for Passing Connection Properties on the JDBC URL

The following sample code passes a libref property on the JDBC URL to a SAS Workspace Server. Here are the scenario details:

- Because you are accessing a workspace server, you must use the IOM driver. The driver class name for the IOM driver is com.sas.rio.MVADriver.
- The host name for the server is c123.na.abc.com, and the port number is 5671.
- The librefs connection property is used to assign a SAS library, which is **`c:\sasdata`**. The first backslash is used to escape the backslash in the actual location.
- Because the server is password-protected, you specify values for the user and password connection properties.

```
import java.sql.*;
public class AccessBase
{
  public static void main(String argv[])
  {
    Connection connection;
```

```
    int i;
    Statement statement;
    String queryString = "SELECT sup_id, sup_name " +
        "FROM mySASLib.suppliers ORDER BY sup_name";
    ResultSet result;
    double id;
    String name;

    try{
      //CONNECT TO THE SERVER BY USING A JDBC URL
      Class.forName("com.sas.rio.MVADriver");
      String user = "jdoe";
      String password = "4ht8d";
      connection = DriverManager.getConnection(
         "jdbc:sasiom://c123.na.abc.com:5671?" +
         "librefs=MySasLib 'c:\\sasdata'",
          user, password);

      //ACCESS DATA
      statement = connection.createStatement();
      result = statement.executeQuery(queryString);
      while (result.next()) {
        id = result.getDouble(1);
        name = result.getString(2);
        System.out.println(id + " " + name);
      }
      statement.close();
      connection.close();
    }
    catch(Exception e){
      System.out.println("error " + e);
    }
  }
}
```

Chapter 4

Connection Recipes

Accessing Specific Types of Data

This chapter explains how to write Java applications that access Base SAS data sets, SPD Server data, and third-party DBMS data. These tasks were chosen because they have some aspects that are specific to SAS. To access the result set that is returned by a query, you use standard JDBC syntax.

For more information about the connection properties used in the sample code, see "Connection Properties for the SAS Drivers" on page 8.

For information about how to control specific aspects of data access, such as limiting the number of retrieved records, see "Working with Data" on page 23.

Accessing DBMS and SPD Server Data

Goal

You want your application to access the following data:

- third-party relational DBMS data that is hosted on a SAS Workspace Server
- SPD Server data that is hosted on a SAS/SHARE server

Implementation

Requirements

Your Java application must meet the following requirements:

- Use the standard JDBC data-access calls.
- Specify the appropriate JDBC driver class name.
- Specify the appropriate host server user ID and host server password.
- Specify the subprotocol and subname for the driver that you are using.
- Use the appropriate **dbms** and **dbmsOptions** connection properties to specify the connection information.
- Use SQL that is specific to the underlying DBMS or the SPD Server.

What Are SAS Engines?

An engine is a component of SAS software that is used to read from or write to a source of data. There are several types of SAS engines, including engines that SAS/ACCESS software uses to connect to a variety of data sources other than Base SAS. Specifically, this recipe discusses how to use SAS/ACCESS engines to access data that is stored in a third-party DBMS or on an SPD Server.

In order to access the DBMS or SPD Server, the JDBC driver uses the **dbms** connection property. The **dbms** connection property specifies the name of the engine that is used to connect directly to the underlying DBMS or the SPD Server. Connecting through an engine enables you to use the SQL syntax of the underlying DBMS or SPD Server.

Valid **dbms** values include **oracle** and **db2** for use with the IOM driver and **spds** for use with the SAS/SHARE driver.

After you use the **dbms** property to specify a DBMS engine or the SPD Server engine, you use the **dbmsOptions** property to specify the engine-specific options. Valid **dbmsOptions** include **user**, **path**, and **dsn** for DBMS engines, and **host** and **serv** for the SPD Server engine.

Note: The JDBC drivers do not process the **dbms** and **dbmsOptions** values. The drivers simply pass the values down to the server.

More about Engine-Specific Options

For a DBMS, engine-specific options are usually the same as the *database-connection-arguments* for the CONNECT statement, which is part of the SAS/ACCESS syntax for the Pass-Through Facility for relational databases. For more information, see "DBMS-Specific Reference" in *SAS/ACCESS for Relational Databases: Reference*.

For information about SPD Server options, see the *SAS Scalable Performance Data Server: User's Guide*.

Assigning Librefs to a DBMS or SPD Server

You can also assign a libref to a DBMS or the SPD Server. However, if you use a libref, then the connection is implicit. An implicit connection uses SAS SQL syntax instead of the underlying SQL syntax. In addition, a connection that is made by using a libref does not support all of the SQL procedure functionality.

Sample Code for Accessing an Oracle DBMS on a SAS Workspace Server

In this example, you are accessing an Oracle DBMS that is hosted on a SAS Workspace Server. Here are the scenario details:

- Because you are accessing a SAS Workspace Server, you must use the IOM driver. The driver class name for the IOM driver is com.sas.rio.MVADriver.
- Because the workspace server is password-protected, you specify values for the **`user`** and **`password`** connection properties. These credentials are authenticated to SAS, as opposed to credentials authenticated to the DBMS.
- You identify the Oracle DBMS by specifying the appropriate values for the **`dbms`** and the **`dbmsOptions`** connection properties. The **`dbmsOptions`** property is the object to use for identifying the user and password credentials authenticated by the DBMS.
- The host name for the workspace server is c123.na.abc.com, and the port number is 8591.

```
import java.sql.*;
import java.util.Properties;

public class iomAccessDbms
{
  public static void main(String argv[])
  {
    Connection connection;
    Properties props;
    int i;
    Statement statement;
    String queryString = "SELECT sup_id, sup_name FROM suppliers ORDER BY sup_name";
    ResultSet result;
    double id;
    String name;

    try {
      //CONNECT TO THE SERVER BY USING A CONNECTION PROPERTY LIST
      Class.forName("com.sas.rio.MVADriver");
      props = new Properties();
      props.setProperty("user","oracleUserJdoe");  /* user acct for SAS */
      props.setProperty("password","oracle2ytr8"); /* passwd for SAS acct */
      props.setProperty("dbms", "ORACLE");
      props.setProperty("dbmsOptions","user='oracleAdminRsmith' " +
         "password='admin3yr8' path='oraclev7'");

      connection = DriverManager.getConnection(
          "jdbc:sasiom://c123.na.abc.com:8591",props);

      //ACCESS DATA
      statement = connection.createStatement();
      result = statement.executeQuery(queryString);
      while (result.next()){
        id = result.getDouble(1);
        name = result.getString(2);
        System.out.println(id + " " + name);
      }
      statement.close();
```

```
      connection.close();
    }
    catch(Exception e){
      System.out.println("error " + e);
    }
  }
}
```

You can also use the JDBC URL to specify connection properties as shown in the following example. Note that the credentials oracleUserJdoe and oracle2ytr8 are authenticated by SAS. The credentials in the **dbmsOptions** property are authenticated by the database.

```
connection = DriverManager.getConnection(
  "jdbc:sasiom://c123.na.abc.com:8591?user=oracleUserJdoe&password=oracle2ytr8" +
  "&dbms=ORACLE&dbmsOptions="user='oracleAdminRsmith' password='admin3yr8' " +
  "path='oraclev7'"
```

Sample Code for Accessing SQL Server and DB2 Data on a SAS Workspace Server

The following table contains sample connection code for using the IOM driver to access other third-party relational DBMS data. User-supplied values such as passwords and user IDs are italicized. Note that SAS/ACCESS must be configured on the server that receives the request.

Table 4.1 *Sample Connection Code for Using the IOM Driver to Access DBMS Data*

Data Source	Sample Connection Code

OLE DB Connection to Microsoft SQL Server	`properties.setProperty("dbms","oledb");`
	`properties.setProperty("dbmsOptions","provider='`*`sqloledb`*`' user='`*`dbadmin`*`'`
	`password='`*`password1`*`' datasource='`*`sql1`*`'");`

DB2	`properties.setProperty("dbms","DB2");`
	`properties.setProperty("dbmsOptions","user='`*`dbadmin`*`'`
	`password='`*`password1`*`' dsn='`*`myDSN`*`'");`

Sample Code for Accessing SPD Server Data on a SAS/SHARE Server

Only the SAS/SHARE driver for JDBC supports access to SPD Server data via the **dbms** and **dbmsOptions** connection properties. The driver class name for the SAS/SHARE driver for JDBC is com.sas.net.sharenet.ShareNetDriver. You can use the following sample connection code:

```
properties.setProperty("dbms", "SPDS");
properties.setProperty("dbmsOptions", "DBQ='qabig1' HOST='xyz.abc.def.com' " +
    "SERV='5190'");
```

Accessing a Base SAS Data Set

Goal

You want to access a Base SAS data set that is hosted on a SAS Workspace or SAS/SHARE server.

Implementation

Requirements

Your Java application must meet the following requirements:

- Use the standard JDBC data-access calls.
- Specify the appropriate JDBC driver class name.
- Specify the appropriate host server user ID and host server password.
- Specify the subprotocol and subname for the driver that you are using.
- Use the librefs connection property in order to assign a SAS library.
- Prepend table names with the libref that identifies the SAS library.

What Are SAS Libraries?

SAS organizes tables in libraries. Before you can use a SAS library, you must tell SAS where it is. One way to identify the library is to use a libref, which is a short name (or alias) for the full physical name of the library. In the SAS programming language, you use a LIBNAME statement in order to define the libref. In your Java application, you use the librefs connection property in order to assign the SAS library.

For example, you have a Base SAS data set named **`suppliers.sas7bdat`** that is stored in a directory named **`c:\sasdata`** on a SAS Workspace Server. The following line of code shows how you set the librefs connection property in order to define mySasLib.

```
props.setProperty("librefs", "mySasLib 'c:\\sasdata'");
```

Often, librefs are already defined for SAS servers, which means you do not need to set the librefs property.

You write your Java application using standard SQL. However, you must prepend a libref to the name of the table that you want to access. For example, after you define mySasLib, you can issue the following query in order to access the data in **`suppliers.sas7bdat`**.

```
String queryString = "SELECT * FROM mySasLib.suppliers";
```

For more information about SAS libraries, librefs, and the LIBNAME statement, see *SAS Language Reference: Dictionary*.

Note: For information about how to access a Base SAS data set without using a LIBNAME statement, see “SAS Libraries: Tools for Managing Libraries” in *SAS Language Reference: Concepts*.

Sample Code for Accessing a Base SAS Data Set

In this example, you use the IOM driver to access a Base SAS data set in a library named mySasLib. Here are the scenario details:

- Because you are accessing a SAS Workspace Server, you must use the IOM driver. The driver class name for the IOM driver is com.sas.rio.MVADriver.
- Because the workspace server is password-protected, you specify values for the user and password connection properties.
- The librefs connection property is used to assign the SAS library located in **`c:\sasdata`**.
- The host name for the workspace server is c123.na.abc.com, and the port number is 8591.

TIP To access the same data set on a SAS/SHARE server, use the SAS/SHARE driver and modify the code to use **`com.sas.net.sharenet.ShareNetDriver`** as the driver class name. Set **`sharenet`** as the URL subprotocol name, and the correct port number for the server.

```
import java.sql.*;
import java.util.Properties;

public class accessSASData
{

  public static void main(String argv[])
  {
    Connection connection;
    Properties props;
    int i;
    Statement statement;
    String queryString = "SELECT sup_id, sup_name " +
        "FROM mySasLib.suppliers ORDER BY sup_name";
    ResultSet result;
    double id;
    String name;

    try {
      //CONNECT TO THE SERVER BY USING A CONNECTION PROPERTY LIST
      Class.forName("com.sas.rio.MVADriver");
      props = new Properties();
      props.setProperty("user","jdoe");
      props.setProperty("password", "4ht8d");
      props.setProperty("librefs", "mySasLib c:\\sasdata';");

      connection = DriverManager.getConnection(
          "jdbc:sasiom://c123.na.abc.com:8591", props);

      //ACCESS DATA
      statement = connection.createStatement();
      result = statement.executeQuery(queryString);
      while (result.next()){
        id = result.getDouble(1);
        name = result.getString(2);
        System.out.println(id + " " + name);
      }
      statement.close();
```

```
        connection.close();
      }
      catch(Exception e){
        System.out.println("error " + e);
      }
    }
  }
```

You can also use the JDBC URL to specify connection properties as shown in the following example:

```
connection = DriverManager.getConnection(
     "jdbc:sasiom://c123.na.abc.com:8591?user=jdoe&password=4ht8d" +
     "&librefs=MySasLib 'c:\\sasdata'");
```

TIP Separate multiple libraries with a semicolon and enclose the paths within parentheses, as shown here:

```
properties.setProperty("librefs",
    '(mySasLib 'c:\\libraryFolder';mySasLib2 'c:\\libraryFolder2')");
```

Using SSPI for Authentication to Access a SAS Workspace Server

Goal

You want to use SSPI for an Integrated Windows Authentication (IWA) connection from a client program running on a Windows host to a SAS Workspace Server running on Windows. A program that uses SSPI does not store credentials in the source code or transmit credentials over the network. The Windows identity of the user account that is running the client program is propagated to the SAS server that the client program connects to. SSPI is available only when the client host and the server are both Windows hosts in the same domain or in domains that trust each other.

Implementation

Requirements

1. Ensure that the SAS ObjectSpawner for the workspace server is configured with the -sspi option. Check the ObjectSpawner.bat file for this option. For more information about the -sspi option, see "Spawner Invocation Options" in *SAS Intelligence Platform: Application Server Administration Guide*.
2. Make the sspiauth.dll file available to the JVM that is running the client program. An sspiauth.dll is available with a SAS Foundation installation such as **`C:\Program Files\SASHome\SASFoundation\9.3\core\sasext`**. This file must be available to the JVM within the java.library.path system property. One method of providing access to the library is to have the dll in the same directory that the JVM will run. Another method is to ensure that the file is available in the PATH environment variable.

Sample Code

In this example, the IOM driver is used to return the number of observations stored in the SASHELP shoes SAS data set.

```
import java.sql.*;
import java.util.Properties;

public class SSPIAuthentication
{

  public static void main(String argv[])
  {
    Connection connection;
    Properties props;
    Statement statement;
    String queryString = "SELECT COUNT(*) " +
        "FROM sashelp.shoes";
    ResultSet result;
    double rowcount;

    try {
      //CONNECT TO THE SERVER BY USING A CONNECTION PROPERTY LIST
      Class.forName("com.sas.rio.MVADriver");
      props = new Properties();
      props.setProperty("usesspi","negotiate");
      /*
       * properties user and password are not specified because
       * SSPI is providing the identity to the workspace server
       */

      connection = DriverManager.getConnection(
          "jdbc:sasiom://c123.na.abc.com:8591", props);

      //ACCESS DATA
      statement = connection.createStatement();
      result = statement.executeQuery(queryString);
      while (result.next()){
        rowcount = result.getDouble(1);
        System.out.println("Number of obs in sashelp.shoes: " + rowcount);
      }
      statement.close();
      connection.close();
    }
    catch(Exception e){
      System.out.println("error " + e);
    }
  }
}
```

Chapter 5

Data Management Recipes

Working with Data

This chapter explains how to perform tasks related to limiting retrieved data, managing missing values, and applying formats. These tasks were chosen because they have some aspects that are specific to SAS.

To perform a task that is not included in this chapter, use Oracle JDBC API documentation along with the connection recipes in "Accessing Specific Types of Data" on page 15.

For more information about the connection properties used in the sample code, see "Connection Properties for the SAS Drivers" on page 8.

Setting the Fetch Size

Goal

You want your application to generate a result set for optimal performance and memory usage.

Implementation

For the IOM driver, the default fetch size is calculated dynamically unless the fetch size is set manually with the setFetchSize method. The getFetchSize method returns 0 to indicate the dynamic sizing. The calculation is performed at run time by determining the width of a row in the result set, and then determining how many rows can be stored in a 16-kilobyte buffer.

This value works well in most cases, and also takes into consideration data sources that have either narrow rows (few columns or short CHAR columns) or wide rows (many columns or long CHAR columns). For data sources with narrow rows, the result set can hold many rows to improve performance. For data sources with wide rows, the result set holds fewer rows so that memory consumption does not reduce performance. Use the setFetchSize method to override this behavior.

The SAS/SHARE driver uses a default fetch size of 10 rows. This value can be changed with the setFetchSize method, but this driver does not perform a dynamic calculation for the optimal fetch size.

Limiting the Number of Retrieved Records

Goal

You want your application to limit the number of records that are retrieved from a data source.

Implementation

To limit the size of a result set, you can use either SAS syntax or standard JDBC syntax. Because there are implementation differences between the two types of syntax, you should use the syntax that best suits your needs.

The sample code in the table can be used to produce a result set that contains a maximum of 20 rows.

Table 5.1 *Two Ways to Limit a Result Set*

Method	Sample Code	Action
SAS syntax*	`stmt.executeUpdate("reset outobs=20")`	• Limits the number of rows that the server sends back. • The limit remains for the duration of the connection, as long as the limit is not changed with subsequent code.

Method	Sample Code	Action
Standard JDBC syntax	`stmt.setMaxRows(20)`	• Limits the number of rows that the client asks for. • The limit remains for the duration of the statement, as long as the limit is not changed with subsequent code.

* Simple queries of the form SELECT * FROM TABLE are not affected by the OUTOBS= option. You must explicitly select the columns or provide a WHERE clause.

Note: In addition, standard JDBC syntax is more portable than SAS syntax.

Reading and Writing NULL and Missing Values

Goal

You want to read and write null and missing values.

Note: SAS uses libraries to organize collections of tables. In most cases, when a table name is used in an SQL string, the table name must be prefixed by a library name. If the SAS Work library is being used, the prefix is not needed. The examples in this recipe assume that the Work library is being used. For more information about SAS libraries, see "What Are SAS Libraries?" on page 19.

Implementation

What Are NULL and Missing Values?

The SAS/SHARE server and the SAS Workspace Server do not explicitly support ANSI SQL NULL values. Instead, they support a SAS concept called missing values. Although ANSI SQL NULL values are sometimes equivalent to SAS missing values, the two concepts are not exactly the same. Here are some of the differences:

- Users can specify many types of SAS missing values for numeric data. An ANSI SQL NULL represents nonexistent data in only one way.
- If a NULL value is written to a character type column, both the SAS/SHARE driver and the IOM driver interpret the value as a SAS missing value and return **`FALSE`** for wasNull.

  Note: If a numeric type is set to NULL, wasNull returns **`TRUE`** as expected.

Note: For more information about SAS missing values, see "Definition of Missing Values" in *SAS Language Reference: Concepts*.

Writing SQL NULL Values

The following code creates a sample table named **`books`** and inserts two SQL NULL values. The SAS/SHARE server and the SAS Workspace Server store these types of values as SAS missing values.

```
stmt.executeUpdate("CREATE TABLE books (c1 varchar(32), c2 double precision)");
stmt.executeUpdate("INSERT INTO books VALUES(null, null)");
```

The following code stores SQL NULL values by using a prepared statement.

```
stmt.executeUpdate("CREATE TABLE books (c1 varchar(32), c2 double precision)");
PreparedStatement ps = connection.prepareStatement("INSERT INTO books VALUES(?,?)");
ps.setNull(1, Types.VARCHAR);
ps.setNull(2, Types.DOUBLE);
ps.executeUpdate();
```

Writing SAS Missing Values

The following code stores SAS missing values. Note that there is little reason to write missing values to character columns because writing null values is more portable. Writing missing values to numeric type columns in SAS data sets might be appropriate if the data sets are used for reporting and the type of missing value is important.

```
stmt.executeUpdate("CREATE TABLE books (c1 varchar(32), c2 double)");
stmt.executeUpdate("INSERT INTO books VALUES(' ', .a)"); /* NOTE: just .a */
```

The following code stores missing values by using a prepared statement.

```
stmt.executeUpdate("CREATE TABLE books (c1 varchar(32), c2 double)");
PreparedStatement ps = connection.prepareStatement("INSERT INTO books VALUES(?, ?");
ps.setString(1, " ");
ps.setObject(2, ".a", Types.DOUBLE);
ps.executeUpdate();
```

Missing character values are represented by a single blank. Numeric values can have special missing value representations. A special missing value enables you to represent different categories of missing data by using the letters A-Z or an underscore.

Note: For more information about special missing values, see "Special Missing Values" in *SAS Language Reference: Concepts*.

Reading Null and Missing Values

The following table describes the values that are returned by the SAS drivers depending on the server, column type, and call.

Table 5.2 *Values That Are Returned by the SAS Drivers for JDBC Depending on the Server, Column Type, and Call*

Call	SAS/SHARE Driver	IOM Driver
	Strings stored as NULL	
getString, getObject	" "*	" "*
wasNull	false**	false**
	Strings stored as missing	
getString, getObject	" "	" "
wasNull	false	false
	Numeric stored as NULL	
getString, getObject	null	null

Call	SAS/SHARE Driver	IOM Driver
getDouble, getFloat, getInt, and so on	0.0	0.0
wasNull	true	true
Numeric stored as missing		
getString, getObject	null	null
getDouble, getFloat, getInt, and so on	0.0	0.0
wasNull	true	true

* The SAS/SHARE and IOM drivers return the nonstandard empty string " ". This empty string is padded with spaces to the size of the column width.

** The SAS/SHARE and IOM drivers return false. Some users might not expect false as a return value.

Missing Values and SAS SQL

As indicated in the previous sections, the Java method wasNull returns true against any numeric type column that holds an ANSI SQL NULL value or a SAS missing value. Also indicated in the previous sections, wasNull unexpectedly returns false against character type columns when reading SAS data sets. If there is a need to detect null values in character columns under these conditions, you can use the SAS SQL "IS [NOT] MISSING" extension to the SQL language or you can trim the value and test against " ".

```
create table books (c1 varchar(32), c2 double precision);
insert into books values (null, null);
select count(*) from books where trim(c1) = " "; /* returns 1 */
select count(*) from books where c1 is missing;  /* also returns 1 */
```

Retrieving Formatted Data

Goal

You want your application to retrieve data from a SAS Workspace Server with formatting applied as it is stored in the Base SAS data set.

Note: The SAS/SHARE driver does not support a connection property that controls formats. However, the drivers do support formatting in the SQL string.

Implementation

Sample Code for Retrieving Formatted Data

Base SAS data sets can have formats associated with each column. When reading SAS data sets with the IOM driver, you can control whether those formats are applied.

By default, the formats associated with a Base SAS data set are not applied when data is retrieved. In order to apply SAS formatting, you set the apply_formats connection property to **`true`**. The code looks like this:

```
properties.setProperty("apply_formats", "true");
```

When apply_formats is set to **true**, all data is returned from the server as a string with column formatting applied. To access the data, you use the getString method on the result set. For example, assume that a column with the format **"5."** has a value of **15.2854**. If apply_formats is set to **true**, then calling getString for that value will return **"15"**.

How Formats Are Determined Regardless of the Property Setting

Regardless of the apply_formats setting, the formats that are associated with a SAS data set determine what column type is returned by the driver.

When a ResultSet is created from a query, the driver obtains column metadata information from SAS. From this metadata, the driver uses parts of the column format to make a type determination. For example, if the format is "DATEx." or "DATETIME", then the type is considered to be an SQL DATE (or TIMESTAMP). If the format is DOLLAR12., then the type is considered to be SQL INT.

Chapter 6

Tips and Best Practices

How to Generate a List of Supported Connection Properties

You can use connection properties in order to control the behavior of a SAS driver. To determine which connection properties are supported by a specific SAS driver for JDBC, you write code that iterates through the DriverPropertyInfo that is returned from the Driver.getPropertyInfo method. The following sample code shows how this task is done.

```
String url = "jdbc:subprotocol://hostName:port";
Driver driver = null;
DriverPropertyInfo [] driverProperties;
driver = DriverManager.getDriver(url);
driverProperties = driver.getPropertyInfo(url, new Properties());
for (i = 0; i < driverProperties.length; i++) {
  System.out.println(driverProperties[i].name);
}
```

Note: For information about the subprotocol, hostName, and port, see "Passing Connection Properties on the JDBC URL" on page 12.

Using Timestamps, Dates, and Times

Understanding SAS Dates

When you compare date, time, and datetime values in SAS data sets, you must consider the following:

- A SAS time value is the number of seconds since the current day began. That is, 0 is 00:00:00 or 12:00:00 a.m., and 86399 is 11:59:59 p.m.
- The drivers return an error for time values that are less than 0 or greater than 86399 (the last second of the day).
- A SAS date value is the number of days since January 1, 1960, GMT. That is, 0 is 01jan1960, and -1 is 31dec1959. The SAS date value can be read with the getDouble method.
- A Java date value is the number of milliseconds since January 1, 1970, GMT. The millisecond value can be read by calling getDate().getTime().
- A SAS datetime value is the number of seconds since midnight on January 1, 1960. That is, 0 is 01jan1960:00:00:00, and -1 is 31dec1959:11:59:59. The SAS datetime value can be read with the getDouble method.
- A Java timestamp records the number of milliseconds since January 1, 1970, GMT and a separate nanoseconds field. The millisecond value can be read by calling getTimestamp().getTime().

Inserting Timestamps, Dates, and Times

The drivers accept the Date, Time, and Timestamp classes from the java.sql package and the JDBC escape sequences for the same types.

The following code fragment illustrates using the time-related classes in the java.sql package with a PreparedStatement:

```
// stmt is a Statement object created with Connection.createStatement()
stmt.executeUpdate(
  "create table sample.timeexamples (c1 date, c2 time, c3 timestamp)"
);

// Use the time-related classes in the java.sql package
// with a PreparedStatement
java.sql.Date d = new java.sql.Date(System.currentTimeMillis());
java.sql.Time t = new java.sql.Time(System.currentTimeMillis());
java.sql.Timestamp ts = new java.sql.Timestamp(System.currentTimeMillis());

PreparedStatement ps = c.prepareStatement("insert into sample.timeexamples
values (?, ?, ?)");
ps.setDate(1, d);
ps.setTime(2, t);
ps.setTimestamp(3, ts);
ps.executeUpdate();

// Let the classes in the java.sql package perform the
```

```
// JDBC escape sequences
ps.setDate(1, java.sql.Date.valueOf("2010-06-10"));
ps.setTime(2, java.sql.Time.valueOf("09:17:00"));
ps.setTimestamp(3, java.sql.Timestamp.valueOf("2010-06-10 09:17:00"));
ps.executeUpdate();

ps.close();
```

Alternatively, the JDBC escape syntax can be coded manually to insert timestamps (keyword **ts**), dates (keyword **d**), and times (keyword **t**). The following code fragment illustrates using the JDBC escape sequence with timestamp.

```
Timestamp ts = new Timestamp(System.currentTimeMillis());

// By using an escape clause with curly braces and the ts keyword,
// the following code is portable across the SAS Drivers for JDBC.
String myDate = new String ('{ts '" + ts.toString() + "'}");
String query =
    "INSERT INTO work.testtable (index, ts, name) VALUES ("
        + i
        + ", "
        + myDate
        + ", "
        + n
        + ")";
```

Creating Timestamp, Date, and Time Columns

SAS uses double precision numbers to store all numeric data, including time-related values. In order to determine how to parse and present the values, SAS uses formats and informats. When a CREATE TABLE statement is sent to a SAS server, the time-related column definitions are parsed, and the appropriate formats are associated with the column. The following table identifies the formats that are associated with the time-related data types:

Table 6.1 *CREATE TABLE Data Types and SAS Data Types*

CREATE TABLE Data Type Name	Java Data Type	SAS Format
date	java.sql.Date	DATE9.
time	java.sql.Time	TIME8.
timestamp	java.sql.Timestamp	DATETIME19.2

The data in the previous table can be helpful for readers who are familiar with SAS programming to understand how SAS interprets the following SQL statement:

```
create table sample.timeexamples (c1 date, c2, time, c3 timestamp);
```

When a SAS driver parses that statement, it converts the JDBC syntax to SAS format instructions. The previous SQL statement is passed to a SAS server as the following code:

```
create table sample.timeexamples (
  c1  num format=date9.,
  c2  num format=time8.,
  c3  num format=datetime19.2
);
```

Updating Result Sets

The IOM driver is the only SAS driver that supports updating ResultSets. For this driver, you must form the result set in one of the following ways:

- Call the proprietary method MVAStatement.getTable(String libref, String tableName).
- Call Statement.executeQuery with the explicit syntax "select * from *table*" where *table* must be in one of these forms:

  tableName
  With this form, the libref WORK is automatically prefixed to the table name.

  libref.table
  With this form, standard SQL elements such as joins, clauses, and column lists are not supported.

The following code sample shows the syntax that is different from basic JDBC and that is relevant to getting a result set that can be updated:

```
import com.sas.rio.MVAResultSet;
import com.sas.rio.MVAStatement;

/* This feature is available with IOM only. */
Class.forName("com.sas.rio.MVADriver");
connection = DriverManager.getConnection(
    "jdbc:sasiom://hostname:port", props);

stmt = (MVAStatement) connection.createStateent(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);

resultSet = stmt.getTable("libref", "tableName");

/* assume that table has cols Name and Age */
resultSet.absolute(5);
resultSet.updateString("name", "jdoe");
resultSet.updateInt("age", 34);
resultSet.updateRow();
```

Scrolling and Concurrency

The IOM driver supports scrolling by following the standard JDBC syntax for ResultSet.TYPE_SCROLL_INSENSITIVE. The SAS/SHARE driver does not support scrolling.

Note: Requests for ResultSet.TYPE_SCROLL_SENSITIVE are not supported. Those result sets are returned with ResultSet.TYPE_SCROLL_INSENSITIVE.

When you use connection.createStatement, if the underlying data store does not support the requested scrolling type or concurrency, then the JDBC driver sets the scrolling type or concurrency to something that is supported. This JDBC specification requirement means that it is a best practice to check the statement object's values for getResultSetType and getResultSetConcurrency to avoid unexpected results.

Closing Statement and ResultSet Objects

The SAS/SHARE driver can manage a maximum of 64 open statements per connection. To prevent your application from reaching that limit, remember to close Statement and ResultSet objects. In addition, when Statement and ResultSet objects are open, memory remains allocated. Therefore, closing Statement and ResultSet objects also saves memory.

Connection Pooling

About Support for Connection Pooling

The SAS Drivers for JDBC do not implement the JDBC pooling classes or interfaces. The recommended approach is to configure the application server to use pooled connections and to use the SAS driver.

Example: Configuring JBoss

This section provides a high-level and simple overview of the JBoss configuration needed to use the SAS drivers. It is assumed that you have JBoss installed, running, and operational. Note that this is just an example. JBoss is not recommended over other application servers.

Ensure that the JAR files for the SAS driver are located within the **lib** directory for the server:

```
$ pwd
.../jboss/jboss-4.2.0.GA/server/SASServer1
$ ls lib/sas.*.jar
lib/sas.core.jar
lib/sas.svc.connection.jar
```

Edit or create a data source XML file in the **deploy** directory:

```
$ cat deploy/iom-ds.xml
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>sas/jdbc/Iom</jndi-name>
    <connection-url>
      jdbc:sasiom://localhost:8591?librefs=
      HELP 'C:\Program Files\SASHome\SASFoundation\9.3\core\sashelp'
    </connection-url>
```

```
    <driver-class>com.sas.rio.MVADriver</driver-class>
    <user-name>sasdemo</user-name>
    <password>{sas001}cGFzc3dvcmQ=</password>
    <min-pool-size>10</min-pool-size>
    <max-pool-size>50</max-pool-size>
  </local-tx-datasource>
</datasources>
```

Identify the DataSource **`jndi-name`** value as shown in the following example:

```
<%@page contentType="text/html"
  import="java.util.*,javax.naming.*,javax.sql.DataSource,
      java.sql.*,java.net.*,java.io.*"
  %>
  <html>
    <body>
      <p>Count of Subsidiaries by Region and Product</p>

<%
  DataSource ds = null;
  Connection con = null;
  InitialContext ic;
  String query = "SELECT REGION, PRODUCT, COUNT(SUBSIDIARY) " +
    "FROM HELP.shoes GROUP BY REGION, PRODUCT";
  String nbsp = " ";

  try {
    ic = new InitialContext();
    ds = (DataSource) ic.lookup("java:/sas/jdbc/Iom");
    con = ds.getConnection();
    ResultSet rs = con.createStatement().executeQuery(query);
    ResultSetMetaData rsmd = rs.getMetaData();
    int colCount = rsmd.getColumnCount();
    out.println("<TABLE border='1'>");
    out.println("<TR>");
    for (int i = 1; i <= colCount; ++i) {
      out.println("<TH>" + rsmd.getColumnLabel(i) + "</TH>");
    }
    out.println("</TR>");

    String val = null;
    while (rs.next()) {
      out.print("<TR>");
      for (int i = 1; i <= colCount; ++i) {
        val = rs.getString(i);
        if (rs.wasNull()) {
          val = nbsp;
        }
        out.print("<TD>" + val + "</TD>");
      }
      out.println("</TR>");
    }
    out.println("</TABLE>");
    rs.close();
    con.close();
  } catch (Exception e) {
    e.printStackTrace();
```

```
    }
%>
    </body>
  </html>
```

Glossary

API
See application programming interface.

application programming interface
a set of software functions that facilitate communication between applications and other kinds of programs or services. Short form: API.

data set
See SAS data set.

data source name
a persistent identifier that is associated with a data source definition. The data source definition specifies how to locate and access a data source, including any authentication (such as a user name and password) that a user must supply in order to access the data. Short form: DSN.

database management system
a software application that enables you to create and manipulate data that is stored in the form of databases. Short form: DBMS.

DBMS
See database management system.

DSN
See data source name.

Integrated Object Model
the set of distributed object interfaces that make SAS software features available to client applications when SAS is executed as an object server. Short form: IOM.

Integrated Object Model server
See IOM server.

IOM
See Integrated Object Model.

IOM server
a SAS object server that is launched in order to fulfill client requests for IOM services. Short form: IOM server.

JAR file
a Java Archive file. The JAR file format is used for aggregating many files into one file. JAR files have the file extension .jar.

Java
a set of technologies for creating software programs in both stand-alone environments and networked environments, and for running those programs safely. Java is an Oracle Corporation trademark.

Java class
in the Java programming language, a definition of a particular kind of object. A class definition defines instance variables, class variables, and methods. It also specifies the interfaces that the class implements and the immediate superclass of the class.

Java Database Connectivity
See JDBC.

Java Development Kit
See JDK.

JDBC
a standard interface for accessing SQL databases. JDBC provides uniform access to a wide range of relational databases. It also provides a common base on which higher-level tools and interfaces can be built. Short form: JDBC.

JDK
a software development environment that is available from Oracle Corporation. The JDK includes a Java Runtime Environment (JRE), a compiler, a debugger, and other tools for developing Java applets and applications. Short form: JDK.

join
the act of combining data from two or more tables in order to produce a single result table.

missing value
a type of value for a variable that contains no data for a particular row or column. By default, SAS writes a missing numeric value as a single period and a missing character value as a blank space.

parallel I/O
a method of input and output that takes advantage of multiple CPUs and multiple controllers, with multiple disks per controller to read or write data in independent threads.

result set
the set of rows or records that a server or other application returns in response to a query.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

SAS IOM workspace
in the IOM object hierarchy for a SAS Workspace Server, an object that represents a single session in SAS.

SAS Metadata Server
a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

SAS Workspace Server
a SAS IOM server that is launched in order to fulfill client requests for IOM workspaces.

SAS/ACCESS software
a group of software interfaces, each of which makes data from a particular external database management system (DBMS) directly available to SAS, as well as making SAS data directly available to the DBMS.

SAS/SHARE server
the result of an execution of the SERVER procedure, which is part of SAS/SHARE software. A server runs in a separate SAS session that services users' SAS sessions by controlling and executing input and output requests to one or more SAS libraries.

Scalable Performance Data Engine
a SAS engine that is able to deliver data to applications rapidly because it organizes the data into a streamlined file format. Short form: SPD Engine.

SPD Engine
See Scalable Performance Data Engine.

SPD Engine data set
a data set created by the SPD Engine that has up to four component files: one for data, one for metadata, and two for any indexes. The minimum number of component files is two: data and metadata. Data is separated from the metadata for SPD Engine file organization.

Work library
a temporary SAS library that is automatically defined by SAS at the beginning of each SAS session or SAS job. Unless you have specified a User library, any newly created SAS file that has a one- level name will be placed in the Work library by default and will be deleted at the end of the current SAS session or job.

workspace
See SAS IOM workspace.