THE POWER TO KNOW.

SAS® Drivers for Federation Server 4.1

User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2014. *SAS® Drivers for Federation Server 4.1: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® Drivers for Federation Server 4.1: User's Guide

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

May 2014

SAS 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 **support.sas.com/bookstore** or call 1-800-727-3228.

Contents

What's New in SAS® Drivers for Federation Server 4.1

Overview

SAS Drivers for Federation Server contains the following changes and enhancements:

- The JDBC driver package changed from DataFlux to SAS.
- New keywords were added for the **DriverManager.getConnection** method.

JDBC Driver Package Change

The JDBC driver package changed from **"com.dataflux.fs.jdbc.driver.FSJDBCDriver"** to **"com.sas.tkts.TKTSDriver"**.

The following Java statement loads the JDBC Driver:
Class.forName("com.sas.tkts.TKTSDriver");

New Keywords for the DriverManager.getConnection Method

New keywords and support were added for the **DriverManager.getConnection** method:

```
proxylist=proxy-server
default:none
```

```
NULL_BEHAVIOR={ "ansi" | "missing"  | "default"}
default:"default"
```

```
SCHEMA_SCOPE={ "all" | "all"  | "default"}
default:"all"
```

Recommended Reading

- *SAS Federation Server: Administrator's Guide*
- *DataFlux Secure Administrator's Guide*

For a complete list of SAS books, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Book Sales Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Chapter 1

Overview

Using the Drivers

You can configure your applications to use the SAS Drivers for Federation Server (ODBC or JDBC) to access data sources on SAS Federation Servers. The drivers are implemented as application programming interfaces (APIs).

To use a driver, follow these steps:

1. Define a data source on a SAS Federation Server.
2. Create a data source name (DSN) on your client host to connect to the SAS Federation Server data source.
3. Program your client application to use an ODBC or JDBC driver to access the data source.

About Security

The SAS Drivers for Federation Server are installed by default with the product DataFlux Secure. The security product enables you to replace the default encryption algorithm, SAS Proprietary Encryption (SASPROPRIETARY) with 256-bit AES encryption.

Encryption is used for all network traffic to and from SAS Federation Server. You can also encrypt the passwords that are included in your data source names (DSNs) to secure them for storage on disk. The level of encryption that you use for your passwords must match the level of encryption that is configured on your SAS Federation Server.

To learn more about security and encryption, see “Why Encrypt?”.

About ASBATCH

The install package for the SAS Drivers includes the ASBATCH utility. Administrators use ASBATCH to maintain the transactional database on the DataFlux Authentication Server. The use of ASBATCH is described in the *DataFlux Authentication Server Administrator's Guide*.

To maintain security, you should install ASBATCH only as needed by the administrators of the DataFlux Authentication Server.

Chapter 2

Post-Install Configuration

Configure the ODBC Driver for SAS Federation Server

Windows

In the Windows operating environment, no further configuration is needed after you install the ODBC Driver for SAS Federation Server.

UNIX or Linux

The distributions for Linux include pre-installed versions of **unixODBC** that are located in the **/usr/lib** or **/usr/lib64** directories. If the distribution contains release 2.3.1 or 2.3.2 of **unixODBC**, you do not need to build the ODBC libraries. After installation of the SAS Drivers for Federation Server on UNIX or Linux, follow these steps:

1. Download the source code for the unixODBC driver to SAS Federation Server. The source code is located at **http://www.unixodbc.org/download**.
2. Before you create the distribution, read the installation steps located in the **/opt/unixODBC-*release*/INSTALL**. Also read the installation steps that apply to your operating environment, such as **/opt/unixODBC-*release*/README.SOLARIS**.
3. Set environment variables for your operating system using one of the following procedures.

   AIX

   If you are using AIX, set environment variables as follows. You should set the environment variables before creating libraries and programs.

   ```
   export OBJECT_MODE=64
   export CFLAGS="-q64 -DBUILD_REAL_64_BIT_MODE
                  DSIZEOF_LONG=8"
   ```

```
export CC=xlc_r
export CPPFLAGS=$CFLAGS
```

Change to the output directory and issue the following commands:

```
cd '/opt/unixODBC-(release/aix/lib'
ar -X64 -x -v libodbc.a
ar -X64 -x -v libodbccr.a
ar -X64 -x -v libodbcinst.a
ln -s ./libodbc.so.1     ./libodbc.so
ln -s ./libodbccr.so.1   ./libodbccr.so
ln -s ./libodbcinst.so.1 ./libodbcinst.so
```

HP-UX H6I Itanium

For HP-UX H6I Itanium, set environment variables as follows:

```
export CFLAGS="+DD64 -DBUILD_REAL_64_BIT_MODE
               -DSIZEOF_LONG=8"
export CPPFLAGS=$CFLAGS
```

Solaris

For Solaris, set environment variables as follows:

```
export CFLAGS="-m64 -DBUILD_REAL_64_BIT_MODE
              -DSIZEOF_LONG=8"
export CC=cc
export CPPFLAGS=$CFLAGS
```

Linux

For Linux, set environment variables as follows:

```
export CFLAGS="-m64 -DBUILD_REAL_64_BIT_MODE
              -DSIZEOF_LONG=8"
export CPPFLAGS=$CFLAGS
```

4. Set the link order so that you link to the POSIX thread library before you link to the standard C library. Linking in this order ensures that you load all of the libraries that are required by the ODBC Driver for SAS Federation Server. The following example command shows the required loading order:

```
/usr/ccs/bin/ld -o UserApplication -u___exit
      -umain logger.o ODBCClient.o -L /usr/local/lib
      -L /usr/lib -L /opt/unixODBC-2.3.0/h6i/lib -lodbc
      -lpthread -lc
```

Configure the JDBC Driver for SAS Federation Server

Recommendations

The JDBC Driver for SAS Federation Server implements interfaces that conform to the standard JDBC API (Application Programming Interface) from Oracle. The driver is used to connect to various data sources configured on SAS Federation Server.

- JDK version: 1.6

- JRE version 1.6 or later. It is highly recommended that you use the SAS Private JRE that is shipped with the product and available through the SAS Deployment Wizard installation.

Note: Reference the Oracle Java Database Connectivity (JDBC) API documentation for information about JDBC specification standards.

Add JAR Files

After you install the JDBC Driver for SAS Federation Server, add the driver's JAR files to your application's **CLASSPATH** environment variable. An alternative is to add the JAR files to the **-classpath**. The standard list of JAR files required on the **CLASSPATH** is listed below:

```
icu4j.jar
log4j.jar
sas.core.jar
sas.core.nls.jar
sas.nls.collator.jar
sas.oda.tkts.jar
sas.oda.tkts.nls.jar
sas.security.sspi.jar
sas.svc.connection.jar
sas.svc.connection.nla.jar
```

The extra package of JAR files, installed with DataFlux Secure, are required for encryption. See "Encrypt Passwords in DSNs" on page 10 for additional information.

```
sas.rutil.jar
sas.rutil.nls.jar
sastpj.rutil.jar
```

Chapter 3

Creating DSNs for the SAS ODBC Driver

Create ODBC DSNs on Windows

Prerequisites

Before you create the ODBC data source names in the Windows operating environment, complete the following tasks:

- Install the SAS Driver for ODBC on your Windows application host.
- Configure data sources on the SAS Federation Server using administration DDL or SAS Federation Server Manager. For more information, see the *SAS Federation Server Administrator's Guide*.

Note: Installation of the latest release of ODBC does not update existing DSNs to point to the current release. Therefore, delete existing DSNs and re-create them using the current release for ODBC.

Create ODBC DSNs

Follow these steps to create ODBC DSNs on Windows:

1. In Windows, start the ODBC Data Source Administrator, as described in **Open the ODBC Data Source Administrator**.
2. Select the **User DSN** or **System DSN** tab.
3. Click **Add**.
4. In the Create New Data Source dialog box, select the driver **SAS 32-bit Federation Server** or **SAS 64-bit Federation Server**.
5. Click **Finish**.
6. In the dialog box SAS Federation Server ODBC Driver Setup, enter the required options and values. To learn about required options, see "Specify Required Options for ODBC Data Sources".
7. Add or change the values of the Advanced Options section as required by your application. To learn about advanced options, see "Specify Advanced Options for ODBC Data Sources".
8. Select **Test Connection** to confirm the validity of your driver setup values.
9. Click **OK** to complete the setup process.

Note: Installation of ODBC 4.1 does not update existing DSNs to point to the current release. Therefore, delete existing DSNs and re-create them using ODBC release 4.1.

Example Registry Entry

The following example depicts a registry entry for **FedServerDSN1**, which connects to the Federation Server with an Oracle base data source named ORACLE_DSN.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\FedServerDSN1]
"Driver"="C:\Program Files (x86)\SASHOME\bin\dfodbc.dll"
"Description"="" "Server"="myserver.us.orion.com" "Port"="21032" "Protocol"="BRIDGE"
"UID"="SAS\myuser" "FSDSN"="ORACLE_DSN" "Cei"="Win cp1252-latin1"
"PWD"="{SAS002}9C943B705636DF691286BEA34C6547D1" "CONCURR_RO"="false"
"CUR_FO"="false" "PSB"="prepare" "XCODE"="ignore" "FSCONSTR"=""
"TRACEFILE"="client.log" "TRACEFLAGS"="" "TRACELEVEL"="off"
"PARMS_FILE"="dfodbcJournal.log" "PARMS_TRACE"="off"
```

Create ODBC DSNs on UNIX or Linux

Overview

To enable an application to connect to ODBC data sources on SAS Federation Servers, set environment variables on the application host and define data sources in a file.

Set Environment Variables

Set the following environment variables in the shell of your application:

LD_LIBRARY_PATH
: Set or append the location of the unixODBC binaries and also set or append the library for the ODBC driver, in ***install-path*/lib**. The following example command appends the two paths, using typical values:

```
export LD_LIBRARY_PATH=/wire/develop/odbc/2.3.1/lax/lib:/opt/SASHOME/fedclient/lib
```

If the platform provides the unixODBC libraries, set the following environment variable:

```
export LD_LIBRARY_PATH=/usr/lib64:/opt/SASHOME/fedclient/lib
```

ODBCINI
: Set the name and location of the file that defines ODBC data sources on your Federation Servers. Set a value such as the following:**/opt/SASHOME/fedclient/lib/dfodbc.ini**.

SAS SECURE
: Set the library path for SAS Secure:

```
export TKERSA2_LIB_PATH=/opt/SASHOME/fedclient/lib
```

TKPATH
: Set or append the location of the libraries that are used by the ODBC Driver for SAS Federation Server. Set or append a value such as the following: **/opt/SASHOME/fedclient/lib/**.

The library path specified for TKPATH reflects the fact that the ODBC Driver for SAS Federation Server can be used only with that version of the SAS Threaded Kernel library. Applications that use other SAS threaded kernel libraries cannot use the ODBC Driver for SAS Federation Server.

The following example uses the script dfsenv to illustrate how to update the TKPATH and LD_LIBRARY_PATH environment variables. If your application runs in a sh, ksh, or bash shell, then execute the following command to update the environment variables:

```
eval './bin/dfsenv sh'
```

If your application runs in a csh or tcsh shell, then execute the following command:

```
eval './bin/dfsenv csh'
```

Define Data Sources

After you install the ODBC Driver for SAS Federation Server on a UNIX or Linux client host, you create the file that defines ODBC data sources. Begin by opening a new file using the name and location that is specified in the environment variable **ODBCINI**.

To define data sources in the file, refer to the following example, which contains values for a DSN named FS_ORACLE.

Note: To add a comment line, begin the line with a **#** character.

```
[ODBC Data Sources]
[FS_ORACLE]
Description = SAS Federation Server DSN
FSDSN = ORACLE_DSN
Driver = /opt/SASHOME/lib/dfodbc.so
SERVER = testFed01.us.orion.com
PORT = 21032
PROTOCOL = BRIDGE
```

```
UID = appConnect
PWD = 25o7xWd0gJzK
CEI = UTF-8
CONCURR_RO = false
CUR_FO = false
PSB = prepare
XCODE = ignore
```

Encrypt Passwords in DSNs

Why Encrypt?

You encrypt the passwords in your DSNs to protect those values when they are stored on disk. Network connections between your client application and the SAS Federation Server are encrypted by default.

Encryption Tools

To encrypt passwords, use the encryption tool **`dfs_crypt`** (in Windows) or **`dfsadmin crypt`** (in UNIX or Linux). For information about using the encryption tools, refer to *'Utilities for SAS Federation Server'* in the *SAS Federation Server Administrator's Guide*.

Encryption Level

The level of encryption that you use for your passwords must match the level of encryption that is used by the SAS Federation Server. The default encryption level is SAS Proprietary Encryption (SASPROPRIETARY). If you upgrade the level of encryption on the SAS Federation Server, then you need to update your existing DSNs. The passwords need to be encrypted again using the encryption algorithm that is used by your SAS Federation Server.

Entering Passwords into Your Client Application

If your client application requires users to enter passwords to access data sources, then it might be necessary for those users to enclose their password entries in single quotation marks. The following example shows how single quotation marks are required by the Microsoft ODBC Test Tool. In the **`SQLDriverConnect`** statement, the password value for **`inConnectionString`** requires single quotation marks:

```
DSN=ORACLE_GENERIC;UID=Local/dsnadm;
     PWD='{SAS003}1804E6B4DEC1DD52DD9552D39BC82C96AD6E';
```

If quotation marks are expected but not provided, you will receive a parsing error message that references the encrypted password value.

Note: Quotation marks are not used around encrypted password values in DSNs. Quotation marks are not used in the Windows registry, the Windows ODBC Administrator, or in the UNIX or Linux file that defines DSNs (typically dfodbc.ini.)

Specify Required Options for ODBC Data Sources

The following table defines the required values for ODBC data sources. The fields and values apply in the Windows, UNIX, and Linux operating environments.

Table 3.1 *Required Options for Data Sources*

Field	Definition	Example
Data Source Name	The ODBC data source name refers to the collection of information that is used to access a SAS Federation Server data source. The name will likely provide information about the data source and possible options. The maximum length of names is specified in **`sqlext.h`** by **`SQL_MAX_DSN_LENGTH`**. The default length is 32 characters. Names are checked by the **`SQLValidDSN`** function to ensure that the names do not include the following characters: [ ] { } ( ) , ; ? ! @ \	FedSrvDB2
Description	Provides additional information about the data source.	Federation Server DB2
Server Name	The network name of the target host of the SAS Federation Server.	dev003.orion.com
Port Number	The port number that is assigned to the SAS Federation Server.	24141
Encoding or CEI	The character encoding that is used by SAS Federation Server.	Win cp1252-latin1 or UTF-8
User ID	The user name that your application uses to connect to the SAS Federation Server. Specify a domain if necessary. The user name appears in plaintext.	HQNT\JSmith
Password	The password that is associated with the user ID. On Windows, the password is encrypted and stored in the Windows registry. On UNIX or Linux, the password is stored in plaintext or encrypted. To learn how to encrypt passwords, see "Encrypt Passwords in DSNs".	Plaintext: **`72Hken11`** Encrypted: **`{SAS003}1804E6B4DEC1DD52DD9552D39BC82C96AD6E`**
FS DSN	The SAS Federation Server data source name.	DB2_DSN
Locale	The locale used for the session. The default locale is English, United States.	en_US-English_United States

Specify Advanced Options for ODBC Data Sources

Introduction

You can specify advanced options in your ODBC DSNs to configure the SQL cursor, parameter set behavior, and transcode errors.

Advanced Options

The following table defines advanced options for ODBC data sources accessed through SAS Federation Server. These values apply in the Windows, UNIX, and Linux operating environments.

An example of an application that requires advanced options is the OpenOffice application, which requires that you enable the options Cursor Concurrency and Cursor Library. Enabling these options prevents SQLFetchScroll from reporting SQL_ERROR with SQL State of HY106, which indicates that a fetch type is out of range.

Note: In the following table, the Keyword value is the name of the option as it appears in the data source definition.

Table 3.2 *Advanced Options for ODBC Data Sources*

Option	Description
Cursor Concurrency	Values: false or true Default: false Keyword: **CONCURR_RO** Description: This option specifies the allowed values for the **SQL_ATTR_CONCURRENCY** attribute. This attribute is set using the **SQLSetStmtAttr** ODBC API. When this option is disabled, the ODBC driver places no limits on the values for the **SQL_ATTR_CONCURRENCY** attribute. When this option is enabled, the driver limits **SQL_ATTR_CONCURRENCY** to only **SQL_CONCUR_READ_ONLY**. Attempts to set **SQL_ATTR_CONCURRENCY** to a value other than **SQL_CONCUR_READ_ONLY** , such as **SQL_CONCUR_LOCK**, result in an SQL state of HYC00.
Cursor Library	Values: false or true Default: false Keyword: **CUR_FO** Description: This option specifies the allowed values for the **SQL_ATTR_CONCURRENCY** attribute. This attribute is set using the **SQLSetStmtAttr**r ODBC API. When this option is disabled, the ODBC driver places no limits on values for the **SQL_ATTR_ODBC_CURSORS** attribute. When this option is enabled, the driver explicitly sets **SQL_ATTR_ODBC_CURSORS** to **SQL_CUR_USE_IF_NEEDED** just prior to connection.

Option	Description
Parameter Set Behavior	Values: emulate, direct, or prepare Default: prepare Keyword: **PSB** Description: This option specifies the manner in which FedSQL emulates parameterized INSERT, DELETE, and UPDATE statements. emulate: Indicates that FedSQL always emulates parameterized INSERT, DELETE, and UPDATE statements using bulk operations (BulkOps). Parameter arrays should always work, provided that the underlying driver supports BulkOps. However, because the user has the option of setting the **PARAMSET_SIZE** after calling Prepare, some parameterized statements that could have been pushed down are emulated instead. direct: Indicates that FedSQL attempts to support the emulation of parameterized INSERT, DELETE, and UPDATE statements. When parameterized statements are attempted, if the underlying driver cannot support those statements, then those statements fail. prepare: Indicates that FedSQL limits **PARAMSET_SIZE** prior to Prepare. Attempts to set **PARAMSET_SIZE** generates an error. FedSQL determines at prepare-time whether to off-load the parameterized INSERT, DELETE, or UPDATE statement to the underlying driver or to attempt to emulate that statement via BulkOps.
Transcode Errors Behavior	Values: error, warning, or ignore Default: ignore Keyword: **XCODE** Description: This option determines how to report errors to applications when the ODBC Driver for SAS Federation Server is unable to transform character data (Transcode) to or from the data source. These errors typically occur when a character cannot be represented in both the application and the data source. The values specified by this option are used to set the **TKTS_ATTR_XCODE_WARN** attribute using the **TKTSSetStmtAttr** API provided by SAS Federation Server. error: Reports transcode errors at the error level. warning: Reports transcode errors at the warning level. ignore: Does not report transcode errors.

Usage Notes for the ODBC Driver

Changing the Default Current Catalog

The ODBC driver returns all of the schemas and tables that are associated with the current catalog. To access tables in a catalog other than the current catalog, you must specify a different catalog using one of the following methods:

- Specify a current catalog name in the connection string using the **CATALOG=** connection option. For example, **CATALOG=*catalog-name***.

- Set the **SQL_ATTR_CURRENT_CATALOG** connection attribute using the **SQLSetConnectAttr** function of the ODBC API. This attribute can be set before or after connection.

Using Microsoft Master Data Services

When you use Microsoft Excel to access data sources on a SAS Federation Server, be sure to configure your data service as case-insensitive. If your data service is case-sensitive, you might receive an error message that states that you submitted an invalid schema name. The requirement for case-insensitivity applies whenever you use Microsoft Master Data Services to access data sources on a SAS Federation Server.

Chapter 4

Creating DSNs for the SAS JDBC Driver

Java Class Name

The name of the Java class that implements **`java.sql.Driver`** in SAS Federation Server is: **`com.sas.tkts.TKTSDriver`**. Use this class name when registering the driver or when configuring software to use the JDBC Driver for SAS Federation Server.

Specify Properties for the JDBC Driver for SAS Federation Server

Introduction

You can specify properties for the JDBC Driver for SAS Federation Server in a URL or by setting **`java.util.Properties`**.

URL

This is the URL syntax for the JDBC Driver for SAS Federation Server:

```
jdbc:sastkts://fed-server-host:fed-server-port
      ?property-name1=property-value1
      &property-name2=property-value2...
      &property-nameX=property-valueX
```

The following example demonstrates the syntax:

```
jdbc:sastkts://devtest001.orion.com:2171?constring=(DSN=DB2DSN1)
      &userName=local\\user1&password=user1password
```

There is no default value for ***fed-server-host*** or ***fed-server-port***.

To connect to a SAS Federation Server that runs on the same host as your client, use the value **localhost** or **127.0.0.1** for ***fed-server-host***.

In a long JDBC URL, it is easy to confuse the JDBC connection properties and the SAS Federation Server driver's connection options. For example, consider this URL:

```
jdbc:sastkts://localhost:2171?cursorType=TKTS_CUR_USE_DRIVER
      &constring=(DRIVER=FIREBIRD;CATALOG=fbdetail;
      DATABASE='c:\fbdata\fbtest.fdb';)
```

In this case, **cursorType** and **constring** are JDBC connection properties.

If properties are set through the URL and are also set in **java.util.Properties**, the properties in the URL take precedent.

java.util.Properties

To set connection properties in the **java.util.Properties** object, add key-value pairs to the object, and pass the object to **DriverManager.getConnection()** or **Driver.connect()**, or use the **set*()** methods on a data source.

Table 4.1 *Connection Properties*

Property	Definition
constring	Specifies that a connection string is used.
cursorType	Specifies the type of cursor: **TKTS_CUR_USE_TKTS**, **TKTS_CUR_USE_DRIVER**, or **TS_CUR_USE_IF_NEEDED**.
defaultFetchSize	Specifies the default fetch size for the statements created by the connection. The default value is 100.
IDLE_TIMEOUT	Specifies the interval, in seconds, after which an idle pooled statement handle is closed and discarded. The default value is 120.
IS_ALLOCATE_OVER_THE_LIMIT	Determines whether the Statement Pool Manager allocates new statements after the maximum number of statements in the pool has been reached. The default value is TRUE.
IS_POOLED_STATEMENT_ENABLED	Determines whether the Statement Pool Manager is used in the current connection. The default value is FALSE.
IS_RECORD_STATISTICS	Determines whether the Statement Pool Manager displays statistics when the connection is closed. The default value is FALSE.

Property	Definition
MAX_STATEMENT_POOL_SIZE	Specifies the maximum number of statement handles that can be stored in the pool. There is no wait if the pool reaches its maximum size. A request either immediately allocates a new statement handle if **IS_ALLOCATE_OVER_THE_LIMIT** is set to TRUE, or it returns an error. The pool never exceeds the **MAX_STATEMENT_POOL_SIZE**. The Statement Pool Manager disposes of a previously pooled statement handle and replaces it with a new one. The maintenance task is freed to close and discard idle statements. The default value is 100.
NULL_BEHAVIOR	Specifies how to insert missing values. **NULL_BEHAVIOR** controls how the user passes and retrieves null or missing values with the double data type. This property defines the behavior for all of the statements beneath it. The values for **NULL_BEHAVIOR** are as follows: default: Specifies that the JDBC driver does not set the statement attribute to null behavior, and it defaults to what is specified on the server. ANSI: Sets the **"TKTS_NB_MISSING"** statement attribute to **"TKTS_NB_ANSI"** indicating that the client wants to deal with ANSI NULLs only when binding a double. missing: Sets the **"TKTS_NB_MISSING"** statement attribute to **"TKTS_NB_MISSING"** indicating that the client wants to deal with SAS missing values only when binding a double or strings. *Note:* JDBC does not support the retrieval of missing values.
password	The password that accompanies the **userName** used to authenticate the client application for access to SAS Federation Server. The login must exist as a user definition in the appropriate Authentication Server. Note that you might need to supply additional authentication parameters to connect to a relational database through the SAS Federation Server. To encrypt the password, see "Encrypt Passwords in DSNs".
POOL_MAINTENANCE_INTERVAL	Specifies the interval, in seconds, between executions of the statement pool maintenance task. The default value is 2 (seconds).
port	Specifies the port number used to access the data source. The default port is 21032.
proxylist	Specifies a list of one or more proxy servers used to connect to a server. If specifying multiple proxy servers, use a semicolon to separate server names.
SCHEMA_SCOPE	Limits the scope of schemas returned for the **getSchema()** and **getSchemas(CatalogName, SchemaPattern)** methods. **SCHEMA_SCOPE** has two values: 'all' and 'default'. When specified with a value of 'all', all catalogs are returned for these methods. If specified with a value of 'default', only schemas under the default catalog are returned for these methods. If no value is specified, the default is 'all'.
server	Specifies the name of the server that is used to host the data source.

Property	Definition
spn	Sets the service principal name to use when connecting to SAS Federation Server. This property is valid only when the server uses Kerberos or Negotiate.
STATISTICS_OUTPUT_DIRECTORY	Specifies the directory that contains a file of statistic records. The default value is the default JVM temporary directory, as specified in the system property **java.io.tmpdir**. The filename has the following format: **stmtPooling*YYMMDD_HHMMSSmmm_pool-Hash-Code.log***.
userName	The user name of the login that is used to authenticate the client application for access to the SAS Federation Server. The login must exist on the associated Authentication Server. Note that you might need to supply additional authentication parameters to connect to a relational database through the SAS Federation Server.
usesspi	Determines whether the driver uses SSPI authentication. Valid values are none, Kerberos, ntlm, or Negotiate.

Example JDBC Connection Using a DSN

The following example accesses an employee table in a sample database using a DSN:

```
package com.sas.fs.doc.example;

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

public class DocTest {
       static {
              try {
                     Class.forName("com.sas.tkts.TKTSDriver");
              } catch (ClassNotFoundException e) {
                     e.printStackTrace();
              }
       }

  public static void main(String argv[])
    {
       try {

            Properties props = new Properties();

            props.put("user", "fedserveruser");
            props.put("password", "{SAS003}5F5F60A0CEC0B921B503926075B4B6EBD08F");
            props.put("constring", "DSN=MYTRANDSN");

            Connection connection = DriverManager.getConnection(
                        "jdbc:sastkts://mymachine.orion.com:2171", props);

            Statement statement = connection.createStatement();
            ResultSet result = statement.executeQuery("SELECT * FROM EMPLOYEE");
            ResultSetMetaData rsmd = result.getMetaData();
```

```
                int numberOfColumns = rsmd.getColumnCount();

                for (int i = 1; i <= numberOfColumns; i++) {
                      System.out.print(rsmd.getColumnLabel(i) + " ");
                }

                System.out.println("");
                while (result.next()) {
                      for (int i = 1; i <= numberOfColumns; i++) {
                            System.out.print(result.getString(i) + " ");
                      }

                     System.out.println("");
                }
                statement.close();
                connection.close();
          }

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

Using the Statement Pool Manager

About the Statement Pool Manager

The Statement Pool Manager of the JDBC Driver for SAS Federation Server manages a list of named handles that apply to a set of frequently used SQL statements.

To enable and configure the Statement Pool Manager, set JDBC connection properties, as described in "Specify Properties for the JDBC Driver for SAS Federation Server".

You can set properties as Java Virtual Machine (JVM) arguments or as TKTS connection properties. Properties defined at the JVM level are the default for all TKTS connections. To override the JVM properties, applications must define the properties at connection time.

When it is initialized, the Statement Pool Manager starts a pool maintenance task. At intervals specified by the property **`POOL_MAINTENANCE_INTERVAL`**, the maintenance task closes and discards idle statements. Statements are removed if they have been idle for the interval specified by the property **`IDLE_TIMEOUT`**.

To retrieve information about the statement pool, you can specify static methods in the **`StatementPoolManager`** class. For information about the static methods, see "Interact with the Statement Pool Manager".

About Statement Attributes

To use the Statement Pool Manager effectively, it is important to understand which statement attributes will be preserved. The **`ResultSetType`** and **`ResultSetConcurrency`** statement attributes are the only attributes guaranteed to be unaltered. All other statement attributes are reset when you return a statement to the pool. Before executing a pooled statement, each application must set all statement attributes as it would with a newly created statement.

About Statement Handles

Statement pooling improves performance by reducing the time it takes to allocate and prepare statements. When using statement pooling, it is important to use a PREPARE statement with parameters rather than explicit values. For example, an application needs to collect information about worldwide sales by region. The following SQL statement allows statement pooling to improve the performance when collecting this information.

```
select * from WORLD_SALES where region = ?
```

The following SQL statements could be used to collect the same information. However, this approach would consume more resources in the statement pool and be less likely to receive the same level of performance improvement as the prior statement.

```
select * from WORLD_SALES where region = 1
select * from WORLD_SALES where region = 7
select * from WORLD_SALES where region = 9
```

In general, it is better to use parameters rather than explicit values. For complex SQL statements, it might be necessary to use explicit values to provide information to the Data Source optimizer. The Data Source optimizer can then format the request to reduce the number of rows to read while processing the request. This is especially important for requests against tables with a large number of rows.

About the Operational Sequence

When your application requests a statement, the Statement Pool Manager responds as follows:

1. If the statement is in the pool, the manager removes it from the pool and makes it available to your application.
2. If the statement is not in the pool and the pool is not full, the manager provides your application with a new statement handle. When the newly created statement is closed by your application, it is put into the pool.
3. The manager throws an exception **`StatementPoolFullException`** to your application if all of the following conditions are true:
   - The statement is not in the pool.
   - The pool is already full.
   - The property **`IS_ALLOCATE_OVER_THE_LIMIT`** is set to FALSE.
4. The manager provides your application with a new statement handle if either of these sets of conditions are met:
   - The statement is not in the pool, and the statement pool is not full.
   - The statement is not in the pool, the statement pool is already full, and the **`IS_ALLOCATE_OVER_THE_LIMIT`** property is set to TRUE.

Set Properties with TKTS

The following example shows how to set properties for the Statement Pool Manager using TKTS at connection time.

For option definitions, see “Interact with the Statement Pool Manager”.

```
Properties connectionProperties = new Properties();
connectionProperties.put(StatementPoolManager.PROP_IS_POOLED_STATEMENT_ENABLED,
      "true");
connectionProperties.put(StatementPoolManager.PROP_MAX_STATEMENT_POOL_SIZE,
      "100");
```

```
connectionProperties.put(StatementPoolManager.PROP_IS_ALLOCATE_OVER_THE_LIMIT,
      "true");

connectionProperties.put(StatementPoolManager.PROP_POOL_MAINTENANCE_INTERVAL,
      "1");

connectionProperties.put(StatementPoolManager.PROP_IDLE_TIMEOUT, "60");

Connection conn = DriverManager.getConnection("jdbc:sastkts://hostName:2100,
      connectionProperties);
```

Set Properties with JVM Arguments

To set default properties for the Statement Pool Manager using JVM arguments, use the following syntax to add options to your Java command.

For option definitions, see "Interact with the Statement Pool Manager".

```
-DIS_POOLED_STATEMENT_ENABLED=TRUE | FALSE

-DMAX_STATEMENT_POOL_SIZE=number-of-statements

-DIS_ALLOCATE_OVER_THE_LIMIT=TRUE | FALSE

-DPOOL_MAINTENANCE_INTERVAL=time-in-seconds

-DIDLE_TIMEOUT=time-in-seconds

-DIS_RECORD_STATISTICS=TRUE | FALSE

-D STATISTICS_OUTPUT_DIRECTORY=absolute-path
```

Interact with the Statement Pool Manager

Use the following static methods in the class **`StatementPoolManager`** to retrieve information about the statement pool. Each method receives a connection as its input parameter:

`public static long getCurrentPoolSize(Connection conn)`
Returns the current number of statements in the pool.

`public static int getIdleTimeout(Connection conn)`
Returns the interval, in seconds, after which an idle pooled statement handle is closed and discarded. For example:

```
Boolean b = StatementPoolManager.isPooledStatementEnabled(connection);

System.out.println("Is Pooled Statement Enabled   = " + b);
```

`public static int getMaxStatementPoolSize(Connection conn)`
Returns the maximum number of statement handles that can be stored in the pool.

`public static long getPoolFullHits(Connection conn)`
Returns the number of times the pool manager tried to create a new statement but the pool was full.

`public static long getPoolHits(Connection conn)`
Returns the number of times a statement was delivered from the pool.

public static int getPoolMaintenanceInterval(Connection conn)
Returns the interval, in seconds, between executions of the statement pool maintenance task.

public static long getPoolMisses(Connection conn)
Returns the number of times the pool manager did not find an available statement handle in the pool.

public static long getPoolPurges(Connection conn)
Returns the number of times the maintenance task purged a pooled statement.

public static boolean isAllocateOverTheLimit(Connection conn)
Returns a flag indicating if the pool manager will allocate a new statement after the maximum number of statements in the pool has been reached.

public static boolean isPooledStatementEnabled(Connection conn)
Returns a flag indicating if statement pooling is in use for the current connection.

public static boolean isPurgeOnWaitTimeout(Connection conn)
Returns an **IS_PURGE_ON_WAIT_TIMEOUT** flag indicating if the pool maintenance task will purge older statements when a **waitTimeout** is detected.

public static int getPurgeThreshold(Connection conn)
Returns **PURGE_THRESHOLD** in an effort to maintain a minimum number of statements in the pool after maintenance routine is executed.

public static boolean isCacheOverTheLimit(Connection conn)
Returns an **IS_CACHE_OVER_POOL_LIMIT** flag indicating if statements created after the pool limit was reached should be cached or not.

public static int getWaitTimeOut(Connection conn)
Returns **WAIT_TIMEOUT** in seconds. This is the interval that a request for a pooled statement should wait before creating a new one.

public static int getActiveTasks()
Returns the number of pool managers using the **ACTIVE TASKS** monitoring task.

Specify a Proxy Server

You can specify one or more proxy servers for a JDBC connection. The **PROXYLIST** option allows the client to communicate with SAS Federation Server through HTTP configurations. In the event that a JDBC application needs to interact with a server that has been secured and cannot be accessed directly, use the **PROXYLIST** option to specify a proxy for the JDBC connection. Specify the **PROXYLIST** directly on the URL or as a property. This example demonstrates how to specify a **PROXYLIST** in a URL:

```
String url = "jdbc:sastkts://hostname:21032?PROXYLIST=http://myProxy.xyz.com";
DriverManager.getConnection(url);
```

This example specifies a **PROXYLIST** property:

```
String url = "jdbc:sastkts"."//hostname:21032"'
props = new Properties();
props.put("PROXYLIST","http://myProxy.xyz.com");
DriverManager.getConnection(url, props);
```

The default port for the proxy connection is 80. If a port is not specified, the default port is used. To customize the proxy port, add the port number after the proxy hostname, preceded by a colon:

```
props.put("proxylist", "http://myProxy.xyz.com:3061");
```

or:

```
String url = "jdbc:sastkts://hostname:21032?PROXYLIST=http://myProxy.xyz.com:3061";
```

Reading and Writing NULL and Missing Values

Overview

The JDBC Driver for SAS Federation Server allows clients to set the behavior for SQL NULL and SAS missing values with the **NULL_BEHAVIOR** property. **NULL_BEHAVIOR** controls how the user passes and retrieves null and missing values for double data types.

What are NULL and Missing Values?

An **ANSI SQL NULL** value indicates that a particular value is not known. A SAS missing value indicates that no data is stored for the variable in the current observation. There are three types of SAS missing values: numeric, character, and special numeric. Although **ANSI SQL NULL** values are sometimes equivalent to SAS missing values, the two concepts sometimes differ. For example:

- Users can specify many types of SAS missing values for numeric data. **ANSI SQL NULL** represents nonexistent data in one way only.
- If a NULL value is written to a character type column, the driver interprets the value as a SAS missing value and returns **FALSE** for the **wasNull** method.

Note: The **wasNull** method returns true for any numeric type column that holds an SQL NULL or SAS missing value.

For more information about SAS missing values, see *SAS Language Reference: Concepts*.

Configuring the SAS Missing Environment

To insert missing values, the **NULL_BEHAVIOR** property must be set on the JDBC connection. The **NULL BEHAVIOR** connection property has to be set to **MISSING** before a missing value can be inserted. Here is an example:

```
Properties connProperties = new Properties();
connProperties.setProperty("NULL_BEHAVIOR", "missing");
connection = DriverManager.getConnection(connURL, connProperties);
```

See the Connection Properties table for an explanation of the values that are available for the **NULL_BEHAVIOR** property. The values are not case sensitive.

Writing SQL NULL Values

SAS uses libraries to organize collections of tables. In most cases, when a table name is used in an SQL string, the name must be prefixed by a library name. If the SAS Work library is used, the prefix is not needed. The examples below demonstrate how to read and write null and missing values, and assume that the Work library is in use. This example creates a sample table named books and inserts two **SQL NULL** values:

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

This example uses a prepared statement to store **SQL NULL** values:

```
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

If writing to character columns, writing null values is more portable than writing missing values. 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.

Note: You must set the **NULL_BEHAVIOR** connection property to **MISSING** before inserting a missing value.

The following example code stores missing values:

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

The following example code stores missing 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.setString(1, " ");
ps.setObject(2, ".a", Types.DOUBLE);
ps.executeUpdate();
```

Missing character values are represented by a single blank space. 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, a-z, a period, or an underscore. It is important to note that attempts to set double columns to these special values without setting the **NULL_BEHAVIOR** connection property to **MISSING** will produce a Java Language exception.

Chapter 5

JDBC Class and Method Limitations

JDBC Limitations and Restrictions

API Limitations

DatabaseMetaData Methods

During testing, the following JDBC or API limitations were observed.

DatabaseMetaData.getAttributes
SAS Federation Server does not support user-defined types or the API to collect information about attributes. This method returns an empty result set.

DatabaseMetaData.getClientInfoProperties
SAS Federation Server does not support client information properties. This method returns an empty result set.

DatabaseMetaData.getColumns()
Columns 19 through 22 returns **NULL** and column 23 returns an empty string:

```
19 SCOPE_CATLOG String ==>NULL
20 SCOPE_SCHEMA String==>NULL
21 SCOPE_TABLE String==>NULL
22 SOURCE_DATA_TYPE short==>NULL
23 IS AUTO_INCREMENT==>(empty string)
```

DatabaseMetaData.getFunctions(),
DatabaseMetaData.getFunctionColumns()
Returns an empty result set because these methods are not supported by TKTS.

DatabaseMetaData.getSuperTables
The Federation Server does not support Super Tables. This method returns an empty result set.

DatabaseMetaData.getSuperTypes
The Federation Server does not support user-defined types and associated hierarchies. This method returns an empty result set.

DatabaseMetaData.getUDTs

The Federation Server does not support user-defined types and associated hierarchies. This method returns an empty result set.

DatabaseMetaData.getUserName()

When the user identifier is not provided while establishing connection, the SAS Federation Server JDBC driver is unable to provide it for **`DatabaseMetaData getUserName`**.

Here are some examples where you would be able to provide **`userid`**:

```
Properties conProps = new Properties();
  conProps.put("user", myUID);
  conProps.put("password", myPWD);
  conProps.put("constring", myConstring);
  String url = "jdbc:sascloud://testhost";
  Connection tConn = DriverManager.getConnection(url, conProps);

  String url = "jdbc:sascloud://testhost&constring=(DSN=MY_DSN)";
  Connection tConn = DriverManager.getConnection(url,  myUID, nyPWD);
```

The following examples do not provide **`userid`**:

```
 String url =
"jdbc:sascloud://testhost&constring=(DSN=MY_DSN,PWD=XXX,user=userid)";
  Connection tConn = DriverManager.getConnection(url);

  Properties conProps = new Properties();
  conProps.put("constring", "(DSN=MY_DSN,PWD=XXX,user=userid)");
  String url = "jdbc:sascloud://testhost";
  Connection tConn = DriverManager.getConnection(url, conProps);
```

Statement Pooling Conflicts

To prevent conflicts with statement pooling, the JDBC driver does not support the following methods for **`PreparedStatement`** and **`CallableStatement`** classes. The application should use multiple statements rather than updating the SQL text of a prepared statement.

PreparedStatement.*method_name*,
CallableStatement.*method_name*

```
void addBatch(String sql)
ResultSet executeQuery(String query)
int executeUpdate(String query)
int executeUpdate(String query, int autoGeneratedKeys)
int executeUpdate(String query, int[] columnIndexes)
int executeUpdate(String query, String[] columnNames)
boolean execute(String query)
boolean execute(String query, int autoGeneratedKeys)
boolean execute(String query, int[] columnIndexes)
boolean execute(String query, String[] columnNames)
```

Chapter 6

JDBC Best Practices

JDBC Driver for SAS Federation Server

Closing Result Sets, Statements, and Connections

The **`ResultSet`**, **`Statement`**, and **`Connection`** objects can consume quite a bit of memory, so it is strongly recommended that you close these objects to release database resources.

Setting JDBC Fetch Size

Setting a fetch size to a value smaller than the default of 100 might contribute to slow fetch performance. A lower fetch size value results in more server trips, which increase execution times. On the other hand, increasing fetch size could improve performance, but setting it too high could result in an out of memory issue. According to various performance benchmarks, a fetch value of 500 is optimal. Here is an example:

```
PreparedStatement stmt = connection.prepareStatement(query);
int fetchSize = 500;
stmt.setFetchSize(fetchSize);
```

Limiting the Number of Fetched Rows

When working with large amounts of data, it is important to limit the number of rows to fetch. To limit the number of rows retrieved from a database, use the statement interface, **`setMaxRows`**. Here is an example that illustrates this:

```
PreparedStatement stmt = connection.prepareStatement(query);
int maxRows = 100;
stmt.setMaxRows(maxRows);
```

Use JDBC Batch Update for Large Inserts

It is a best practice to insert and update large amounts of data in batches. When performing a large number of parameterized inserts, use JDBC Batch Update. When you submit multiple SQL statements in batch instead of individually, the number of round trips to the database is reduced, which results in significant performance improvement. Here is an example:

```
PreparedStatement pstmt = null;

pstmt = connection.prepareStatement("INSERT INTO \"USERS\"VALUES (?, ?, ?)");

pstmt.setString(1, "john_doe");
pstmt.setInt(2, 25);
pstmt.setString(3, "like to walk");
pstmt.addBatch();

pstmt.setString(1, "john_moe");
pstmt.setInt(2, 50);
pstmt.setString(3, "like to play");
pstmt.addBatch();

pstmt.setString(1, "john_coe");
pstmt.setInt(2, 60);
pstmt.setString(3, "like to eat");
pstmt.addBatch();

pstmt.executeBatch();
```

SAS Federation Server conforms to the following Oracle JDBC rule:

"For any given Statement, an application should not modify the value argument passed to a **`setXXX`** method after the **`setXXX`** method is called and before the subsequent **`execute`**, **`executeQuery`**, **`executeUpdate`**, **`excuteBatch`** or **`clearParameters`** method is called. An application may modify the value argument after the **`execute`**, **`executeQuery`**, **`executeUpdate`**, **`excuteBatch`** or **`clearParameters`** method is called, if there is a subsequent **`setXXX`** method call that overwrites the previous value or if the Statement is not reused. Failure to conform to this restriction may result in unpredictable behavior."

Statement Pooling

Use the statement pooling feature when executing the same statement multiple times. Statement pooling improves performance by reducing processing for the **`allocate`** and **`prepare`** statements. For more information, see "Using the Statement Pool Manager" on page 19.

Invoking last and isLast Methods

Some databases and underlying drivers do not support a scrollable result set. In this case when requesting this type of a result set the following error message within an exception is returned: **`"Unable to create Statement with requested result set concurrency[1008]"`**. Here is an example of this request:

```
try {
```

```
stmt = currentConn.createStatement(
              ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
} catch (SQLException e1) {
       printSQLExceptions(e1, "HY000", UNSUPPORTED_SCROLL_MESSAGE);
}
```

If a scrollable result set is supported, then an application can call **`last()`** and **`isLast()`** methods. Calling **`last()`** moves the cursor to the last row in the result set and calling **`isLast()`** determines whether the cursor is on the last row. Calling **`isLast()`** can impact performance because the JDBC driver might need to fetch ahead one row in order to determine whether the current row is the last row in the result set. If high performance is essential, avoid using the **`isLast()`** method. Here is an example that uses the **`last()`** and **`isLast()`** methods:

```
boolean moved = false;
String selectFromTable = "SELECT * FROM \"EMPLOYEES\"";

try {
       stmt = currentConn.createStatement(
              ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
} catch (SQLException e1) {
       printSQLExceptions(e1);
}

try {
       rs = stmt.executeQuery(selectFromTable);
} catch (SQLException e) {
       printSQLExceptions(e);
}

// move cursor to the last row
try {
       moved = rs.last();
} catch (SQLException e) {
       printSQLExceptions(e);
}

// determine if this is a last row
       try {
              if (rs.isLast()){
                     System.out.println("This is a last row in a result set.");
              }
       } catch (SQLException e) {
              printSQLExceptions(e);
}
```

In a case when a scrollable result set is not supported calling **`last()`** on a **`ResultSet`** object produces the following error: **`"Unable to perform requested operation on a forward only result set"`**. Calling the **`isLast()`** method results in the error: **`"Method not supported when resultSetType is TYPE_FORWARD_ONLY"`**.