Progress DataDirect for JDBC for SAP HANA

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for SAP HANA Driver

The Progress® DataDirect® for JDBC™ for SAP HANA® driver supports read-write access for JDBC applications to the SAP HANA database. The driver provides access to data in SAP HANA that can be pulled into a data visualization tool to get important insights into engineering operations. This functionality allows seamless integration with third-party software and provides comprehensive SQL support and JDBC standard connectivity to BI (Business Intelligence) and ETL (Extract,Transform, Load) tools.

The documentation for the driver also includes the *Progress DataDirect for JDBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for JDBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools.

For the complete documentation set, visit the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/category/datadirect-sap-hana.

For details, see the following topics:

- Troubleshooting
- Additional information
- Contacting Technical Support

What's new in this release?

Support and certification

Visit the following web pages for the latest support and certification information.

- Release Notes: https://www.progress.com/datadirect-connectors/whats-new#jdbc
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since 6.0.0

- The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See FIPS (Federal Information Processing Standard) on page 46 for details.

Highlights of 6.0.0 Release

- The driver supports read-write access to SAP HANA.
- The driver supports JDBC core functions. For details, refer to JDBC support in the Progress DataDirect for JDBC Drivers Reference.
- The driver supports the SAP HANA data types. See Data types on page 13 and getTypeInfo() on page 14 for details.
- The driver supports user ID and password authentication method with SCRAM-SHA-256 mechanism. See Authentication on page 43 and AuthenticationMethod on page 57 for more details.
- The driver provides proxy support. See Proxy server on page 12 and Connection property descriptions on page 49 for more information.
- The driver supports the handling of large result sets with paging and the FetchSize connection property. See FetchSize for more details.
- The driver supports TLS/SSL data encryption. See Connection property descriptions on page 49 for more information.
- The driver supports returning parameter metadata for stored procedure arguments.
- The driver supports catalog API calls.
- The driver supports prepared statements and parameterized queries.
- The driver supports manual transactions for commit and rollback operations. See Driver specifications on page 23 for more information.
- Interactive SQL is now installed with the product. Interactive SQL is a command-line interface that supports connecting your driver to a data source, executing SQL statements and retrieving results in a terminal. This tool provides a method to quickly test your drivers in an environment that does not support GUIs. See Interactive SQL on page 30 for details.

Requirements

The driver is compatible with JDBC 2.0, 3.0, and 4.0.

The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.

Installing and setting up the driver

This section provides you with an overview of the steps required to install and set-up the driver. After completing this procedure, you will be able to begin accessing data with your application.

To begin accessing data with the driver:

1. Install the driver:
   a) After downloading the product, unzip the installer files to a temporary directory.
   b) From the installer directory, run the appropriate installer file to start the installer.
      - **Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.exe`
      - **Non-Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.jar`
   c) Follow the prompts to complete installation.

   The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for JDBC Drivers Installation Guide*.
2. Set your system CLASSPATH to include the driver `.jar` file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. The following examples demonstrate setting the CLASSPATH from a command line using the default installation directory.
   - **Windows Example**

     ```
     CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\saphana.jar
     ```
   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/saphana.jar
     ```
3. Configure your driver using one of the following methods:
   - **Connection URL**: You can begin using the driver immediately by passing a connection URL with your application or tool. The following example show how to connect using user ID and password authentication.

     ```
     jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret;
     ```

     Note: See User ID/password authentication on page 11 for details.
   - **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

Note: For most connections, specifying the minimum required connection properties is sufficient to begin accessing data; however, you can provide values for optional properties to use additional supported features and improve performance.

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:
   - Connection URL Examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
   - Connection Property Descriptions provides a complete list of supported properties by functionality.
   - Performance Considerations describes connection properties that affect performance, along with recommended settings.
5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:
   - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
   - DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.
   - Interactive SQL: Interactive SQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal. This tool provides a method of quickly testing your driver in an environment that does not support GUIs.
   - DataDirect Test: DataDirect Test allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.

This completes the deployment of the driver.

Driver and DataSource classes

The following are the `Driver` and `DataSource` classes used by the driver:

Driver class:

com.ddtek.jdbc.saphana.SAPHanaDriver

DataSource class:

com.ddtek.jdbcx.saphana.SAPHanaDataSource

Connection URL examples

After setting the CLASSPATH, the connection information needs to be passed in the form of a connection URL. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

Note:

- You can also use the DataDirect Configuration Manager tool to generate and test connection URLs. For more information, see "Generating connection URLs with the Configuration Manager."
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

User ID/password authentication

This string includes the properties used to connect with the user ID/password authentication.

```
jdbc:datadirect:saphana://server_name:port_number;User=user_name;
Password=password;[property=value[;...]];
```

where:

server_name

specifies either the IP address or the server name of the primary database server.

port_number

specifies the TCP port of the primary database server that is listening for connections to the database. The default value is `30015`.

user_name

specifies the user ID that is used to connect to the SAP HANA database.

password

specifies a password that is used to connect to your SAP HANA database.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with the user ID/password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret;");
```

Proxy server

This string includes the properties you may need to connect through a proxy server with user ID and password authentication.

```
jdbc:datadirect:saphana://server_name:port_number;ProxyHost=proxy_host;
ProxyPassword=proxy_password;ProxyPort=proxy_port;ProxyUser=proxy_user;
User=user_name;Password=password;[property=value[;...]];
```

where:

server_name

specifies the server name or IP address of the service to which you want to connect. For example, `MyServer`.

port_number

specifies the TCP port of the primary database server that is listening for connections to the database. The default is `30015`.

proxy_host

specifies the proxy server to use for the first connection.

proxy_password

specifies the password needed to connect to a proxy server for the first connection.

proxy_port

specifies the port number where the proxy server is listening for requests for the first connection. The default is `0`.

proxy_user

specifies the user name needed to connect to a proxy server for the first connection.

user_name

specifies the user ID that is used to connect to the SAP HANA instance. For example, `jsmith`.

password

specifies the password used to connect to your SAP HANA instance.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for using a proxy server with user ID and password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:saphana://MyServer:30015;ProxyHost=pserver;
  ProxyPassword=proxysecret;ProxyPort=808;ProxyUser=PUser;User=jsmith;
  Password=secret;");
```

See also

Connection property descriptions on page 49

Data types

The following table lists the data types supported by the driver and describes how they are mapped to JDBC data types. See "getTypeInfo()" for getTypeInfo() results of data types supported by the driver.

Note: The LOB data types only support SELECT operations.

Table 1: SAP HANA Data Types

SAP HANA Data Type	JDBC Data Type
Alphanum	NVARCHAR
Bigint	BIGINT
Binary	BINARY
Bintext	NCLOB
Blob	BLOB
Boolean	BOOLEAN
Char	CHAR
Clob	CLOB
Date	DATE
Decimal	DECIMAL
Double	DOUBLE
Integer	INTEGER
Nchar	NCHAR
Nclob	NCLOB
Nvarchar	NVARCHAR
Real	REAL
Seconddate	TIMESTAMP
Shorttext	NVARCHAR

SAP HANA Data Type	JDBC Data Type
Smalldecimal	DECIMAL
Smallint	SMALLINT
Text	NCLOB
Time	TIME
Timestamp	TIMESTAMP
Tinyint	TINYINT
Varbinary	VARBINARY
Varchar	VARCHAR

See also

getTypeInfo() on page 14

getTypeInfo()

The following table provides getTypeInfo() results for SAP HANA databases supported by the driver.

Table 2: getTypeInfo()

TYPE_NAME = alphanum AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -9 (NVARCHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 127 SEARCHABLE = 3 SQL_DATA_TYPE = -9 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = bigint AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = -5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0
TYPE_NAME = binary AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2000 SEARCHABLE = 3 SQL_DATA_TYPE = -2 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = bintext AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = NULL DATA_TYPE = 2011 (NCLOB) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = -10 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = blob AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = NULL DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = boolean AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 16 (BOOLEAN) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2 SEARCHABLE = 2 SQL_DATA_TYPE = 16 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0
TYPE_NAME = char AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2000 SEARCHABLE = 3 SQL_DATA_TYPE = 1 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = clob AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = NULL DATA_TYPE = 2005 (CLOB) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = 91 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = decimal AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = *precision, scale* DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 34	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 34 SEARCHABLE = 2 SQL_DATA_TYPE = 3 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0

TYPE_NAME = double AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 52 SEARCHABLE = 2 SQL_DATA_TYPE = 8 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0
TYPE_NAME = integer AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = 4 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0
TYPE_NAME = nchar AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -15 (NCHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2000 SEARCHABLE = 3 SQL_DATA_TYPE = -8 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = nclob AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = NULL DATA_TYPE = 2011 (NCLOB) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = nvarchar AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -9 (NVARCHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 5000 SEARCHABLE = 3 SQL_DATA_TYPE = -9 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = real AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 23 SEARCHABLE = 2 SQL_DATA_TYPE = 7 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0

TYPE_NAME = seconddate AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = shorttext AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -9 (NVARCHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 5000 SEARCHABLE = 3 SQL_DATA_TYPE = -9 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = smalldecimal AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 16 SEARCHABLE = 2 SQL_DATA_TYPE = 3000 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0

TYPE_NAME = smallint AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 2 SQL_DATA_TYPE = 5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0
TYPE_NAME = text AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = NULL DATA_TYPE = 2011 (NCLOB) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = -10 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = time AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 2 SQL_DATA_TYPE = 92 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = timestamp AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 7	MINIMUM_SCALE = 7 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 27 SEARCHABLE = 2 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = tinyint AUTO_INCREMENT = 0 CASE_SENSITIVE = 0 CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = 1 LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 2 SQL_DATA_TYPE = -6 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = 0

TYPE_NAME = varbinary AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 5000 SEARCHABLE = 3 SQL_DATA_TYPE = -3 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar AUTO_INCREMENT = 0 CASE_SENSITIVE = 1 CREATE_PARAMS = *length* DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = 0 LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 65535 SEARCHABLE = 3 SQL_DATA_TYPE = 12 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Driver specifications

This section describes the general functionality supported by the driver.

- **Unicode support**: Multilingual JDBC applications can be developed on any operating system using the driver to access both Unicode and non-Unicode enabled databases. Internally, Java applications use UTF-16 Unicode encoding for string data. When fetching data, the driver automatically performs the conversion from the character encoding used by the database to UTF-16. Similarly, when inserting or updating data in the database, the driver automatically converts UTF-16 encoding to the character encoding used by the database.

  The JDBC API provides mechanisms for retrieving and storing character data encoded as Unicode (UTF-16) or ASCII. Additionally, the Java String object contains methods for converting UTF-16 encoding of string data to or from many popular character encodings.

- **Isolation levels**: The driver supports the Read Committed isolation level.

- **Scrollable cursor support**: The driver supports scroll-insensitive result sets.

Note: When the driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more SQLWarnings with detailed information.

- **Error handling**: The driver reports errors to the application by throwing SQLExceptions. Each SQLException contains the following information:
  - Description of the probable cause of the error, prefixed by the component that generated the error
  - Native error code (if applicable)
  - String containing the XOPEN SQLstate

DataDirect tools

Progress DataDirect for JDBC drivers install the set of tools described in this section. For detailed instructions on using these tools, refer to the corresponding topics in the *Progress DataDirect for JDBC Drivers Reference*.

- DataDirect Test allows you to test your JDBC driver and learn the JDBC API.
- DataDirect Connection Pool Manager allows you to pool connections when accessing databases. When your applications use connection pooling, connections are reused rather than created each time a connection is requested. Because establishing a connection is among the most costly operations an application may perform, using Connection Pool Manager to implement connection pooling can significantly improve performance.
- Statement Pool Monitor loads statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- DataDirect Spy logs detailed information about calls your driver makes that can be used for troubleshooting.

Troubleshooting

The *Progress DataDirect for JDBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to the "Troubleshooting" section in the *Reference* for details.

Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for JDBC Drivers Reference* or use the links below to view some common topics:

- "JDBC support" describes support for JDBC interfaces and methods for the Progress DataDirect for JDBC drivers.
- "JDBC extensions" describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package.
- "SQL escape sequences for JDBC" provides an overview of SQL escape sequences for JDBC. In addition, it documents the scalar functions that you use in SQL statements.
- "Security best practices for JDBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

https://www.progress.com/support

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

2

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications. For information on installing your driver and setting the CLASSPATH, see "Installing and setting-up the driver."

For details, see the following topics:

- Tableau
- DbVisualizer
- Interactive SQL

Tableau

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

To use the driver to access data with Tableau:

1. Navigate to the `\lib\xx` subdirectory of the Progress DataDirect installation directory; then, locate the `jar` file for your driver:

   ```
   saphana.jar
   ```

2. Copy the `.jar` file for your driver into the following directory:

Windows: `C:\Program Files\Tableau\Drivers`

Linux: `/opt/tableau/tableau_driver/jdbc`

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.
   - **URL**: Copy and paste your connection URL into this field. The following examples show how to connect using user ID and password authentication.

     ```
     jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret;
     ```

     Note: See User ID/password authentication on page 11 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

You have successfully accessed your data and are now ready to create reports with Tableau. For detailed information, refer to the Tableau product documentation at: https://www.tableau.com/support/help.

DbVisualizer

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with the third-party DbVisualizer tool. The following topics guide you through using DbVisualizer to add your driver, connect, and execute SQL statements.

Adding a driver

To add a driver with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Tools**>**Driver Manager**. The Driver Manager window opens.
3. From the Driver Manager menu, select **Driver**>**Create Driver**.
4. Click the  button to navigate to the location of the driver jar file; then, click **OK**. The following are the default locations for the driver:

   Windows

   ```
   C:\Program Files\Progress\DataDirect\JDBC\lib\60\saphana.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/saphana.jar
   ```

5. Provide values for the following fields; then, close the Driver Manager window.

- **Name**: Type an alias for your driver. For example:

  SAPHANA

- **URL Format**: Optionally, specify the format of the connection string for your driver. For example:

  ```
  jdbc:datadirect:saphana://MyServer:30015
  ```

- **Driver Class**: From the drop down menu, select the driver class for your driver. For example:

  ```
  com.ddtek.jdbc.saphana.SAPHanaDriver
  ```

You can now use your driver with DbVisualizer. Proceed to "Connecting and executing SQL statements" for information on connecting and executing SQL statements.

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Database**>**New Connection**. When prompted to use the Connection Wizard, click **OK**.
3. Provide the following information when prompted; then, click **Next** to proceed:
   - **Connection alias**: Type the name to be used when referring to this connection.
   - **Driver**: Select the alias that you provided for your driver from the drop-down menu.
4. Provide values for the following fields; then, click **Finish**.
   - **Database URL**: Copy and paste your connection URL into this field. The following examples show how to connect using user ID and password authentication.

     ```
     jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret;
     ```

   Note: See Authentication on page 43 for details.

5. To execute SQL statements, select **SQL Commander**>**New SQL Commander**. A SQL Commander tab opens.
6. Select values for the following fields:
   - **Database Connection**: Select connection alias you provided for the connection from the drop-down menu.
   - **Schema**: Select the schema you want to execute queries against from the drop-down menu.
7. In the SQL Commander tab, enter SQL commands you want to execute; then select **SQL Commander**>**Execute**. For example:

   To select all of the rows from the EXAMPLE table:

   ```
   SELECT * FROM EXAMPLE
   ```

To select the URLs for a specified issue:

```
SELECT * FROM EXAMPLE WHERE USER = <ID>
```

See "Supported SQL statements and extensions" for the supported syntax used to execute SQL statements.

Note: If you are fetching large sets of data, you may want to limit the results using the Max Rows and Max Chars fields.

You have successfully accessed your data with DbVisualizer.

Interactive SQL

After you have installed your driver, you can use the driver to access your data with the Interactive SQL tool. Interactive SQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal.

To execute commands with Interactive SQL:

1. Start the ISQL tool. From a command line, enter the following:

   ```
   java -jar saphana.jar --isql
   ```

2. Enter connection properties one at a time by typing *property*=*value*, then pressing **Enter**. For example, to configure the ServerName property:

   ```
   ServerName=//MyServer:30015
   ```

3. After specifying values for your properties, type `connect`, then press **Enter**. If successful, the tool will return a confirmation message.

   Note: If you are unable to connect, you can review the URL by entering the `SHOW URL` command.

4. At the `ISQL>` prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

   ```
   SELECT * FROM EXAMPLE_TABLES;
   ```

   Note: SQL commands must be terminated by a semi-colon.

   Note: In addition to SQL commands, the tool supports a set of proprietary commands. Type `Help` at the prompt for a list of supported commands and syntax.

   The results of the command are displayed in the terminal.

5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

   ```
   DISCONNECT
   ```

6. To end the session, type `exit`; then, press **ENTER**.

3

Configuring and connecting

This section provides information on how to connect to your data store using either the JDBC Driver Manager or DataDirect JDBC data sources, as well as information on how to implement and use functionality supported by the driver.

After the driver has been installed and defined on your classpath, you can connect from your application to your data in either of the following ways.

- Using the JDBC `DriverManager` by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC data source that can be accessed through the Java Naming Directory Interface (JNDI).

For details, see the following topics:

Setting the classpath

The driver must be defined on your CLASSPATH before you can connect. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the driver jar file as shown, where *`install_dir`* is the path to your product installation directory.

```
install_dir/lib/60/saphana.jar
```

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\saphana.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/saphana.jar
```

Connecting using the JDBC Driver Manager

One way to connect to a service is through the JDBC DriverManager using the `DriverManager.getConnection()` method. As the following examples show, this method specifies a string containing a connection URL.

User ID and password authentication

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret;");
```

Note: See User ID/password authentication on page 11 for details.

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with user ID and password authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:saphana://server_name:port_number;User=user_name;
Password=password;[property=value[;...]];
```

where:

`server_name`

specifies either the IP address or the server name of the primary database server.

port_number

specifies the TCP port of the primary database server that is listening for connections to the database.

user_name

Specifies the user ID used to authenticate to SAP HANA with user ID and password authentication method.

password

Specifies the password used to authenticate to SAP HANA with user ID and password authentication method.

Important: The password is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with the user ID and password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret");
```

See also

Connection URL examples on page 10
Connection property descriptions on page 49

Testing the connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

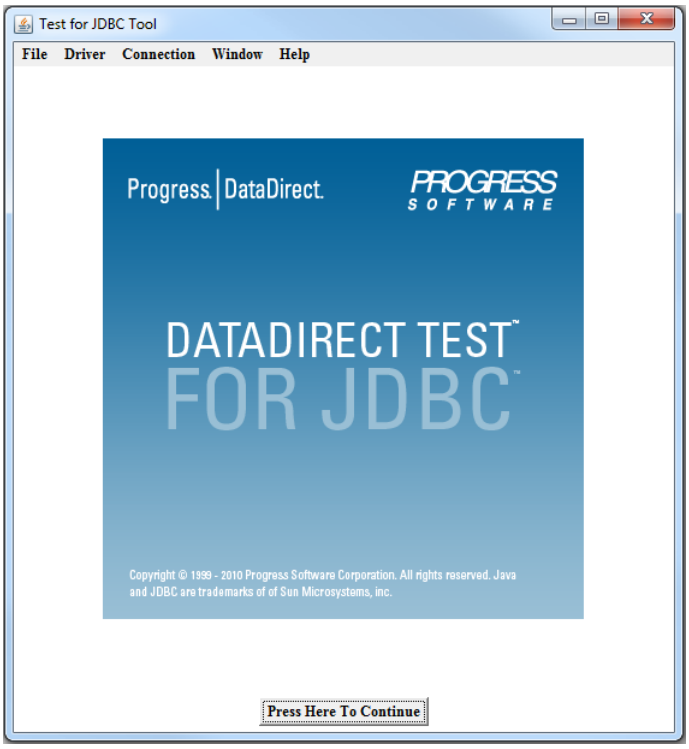
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

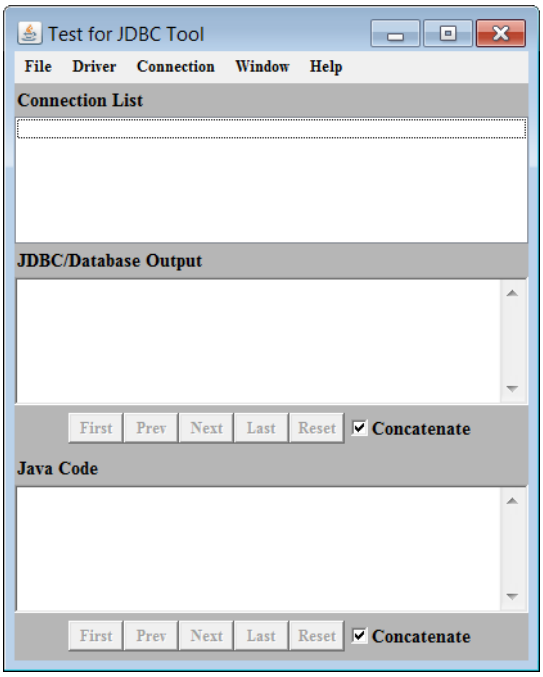
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



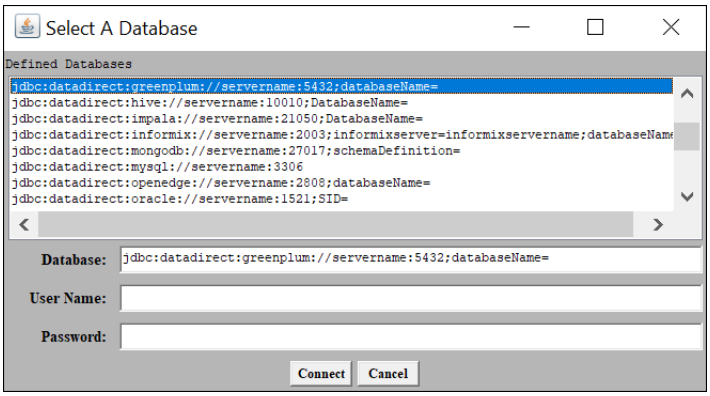
3. Click **Press Here to Continue**.

   The main dialog appears:



4. From the menu bar, select **Connection > Connect to DB**.

   The **Select A Database** dialog appears:



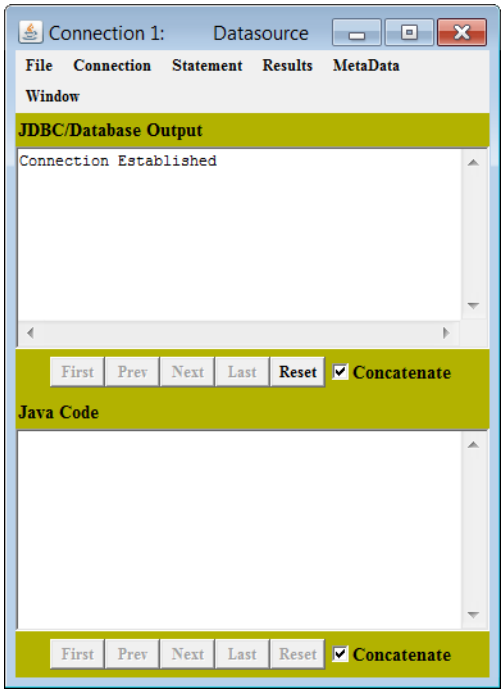
5. Select the appropriate database template from the **Defined Databases** field.
6. In the **Database** field, specify all required connection properties.

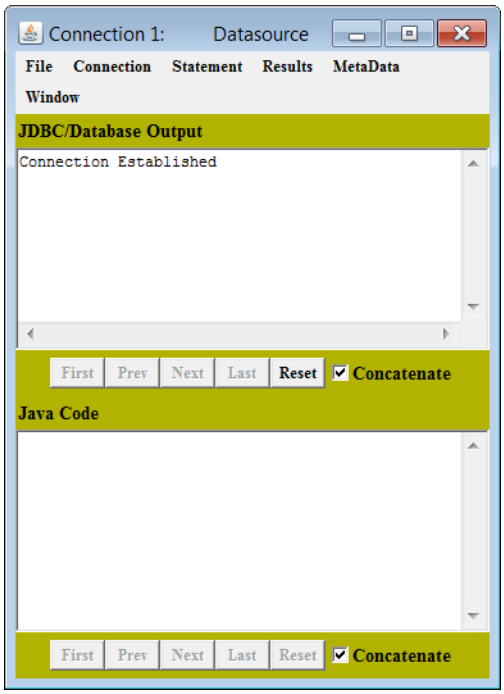
   For example:

   ```
   jdbc:datadirect:saphana://myserver:30015;User=JSmith;Password=secret;
   ```

7. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)





Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

Connecting using data sources

A *JDBC data source* is a Java object, specifically a `DataSource` object, that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a `DataSource` object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the `DataSource` object. The applications using the database do not need to change because they only refer to the name of the data source.

How data sources are implemented

Data sources are implemented through a `DataSource` class. A data source class implements the following interfaces.

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Driver and DataSource classes on page 10

Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *install_dir* is the product installation directory.

- *install_dir*/Examples/JNDI/JNDI_LDAP_Example.java can be used to create a JDBC data source and save it in your LDAP directory using the JNDI Provider for LDAP.
- *install_dir*/Examples/JNDI/JNDI_FILESYSTEM_Example.java can be used to create a JDBC data source and save it in your local file system using the File System JNDI Provider.

See "Example data source" for an example data source definition for the example files.

To connect using a JNDI data source, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the Oracle Technology Network Java SE Support downloads page, unzip the files to an appropriate location, and add the fscontext.jar and providerutil.jar files to your CLASSPATH. These steps are not required for LDAP implementations because the LDAP Service Provider is included with supported versions of Java SE.

Example data source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.saphana.SAPHanaDataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required for a connection using the user ID and password authentication.

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, setUser("abc@defcorp.com").

```
SAPHanaDataSource mds = new SAPHanaDataSource();
mds.setDescription("My SAP HANA Data Source");
mds.setServerName("MyServer");
mds.setPortNumber(30015);
mds.setUser("JSmith");
mds.setPassword ("Password");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for the minimum properties required to establish a connection:

```
if (ds instanceof SAPHanaDataSource)
{
SAPHanaDataSource jmds = (SAPHanaDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
System.out.println("userName=" + jmds.getUser());
System.out.println("password=" + jmds.getPassword());
...
System.out.println();
}
```

Calling a data source in an application

Applications can call a Progress DataDirect data source using a logical name to retrieve the `javax.sql.DataSource` object. This object loads the specified driver and can be used to establish a connection to the database.

Once the data source has been registered with JNDI, it can be used by your JDBC application as shown in the following code example.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (`EmployeeDB`). The `Context.lookup()` method returns a reference to a Java object, which is narrowed to a `javax.sql.DataSource` object. Then, the `DataSource.getConnection()` method is called to establish a connection.

Testing a data source connection

You can use DataDirect Test™ to establish and test a data source connection. The screen shots in this section were taken on a Windows system.

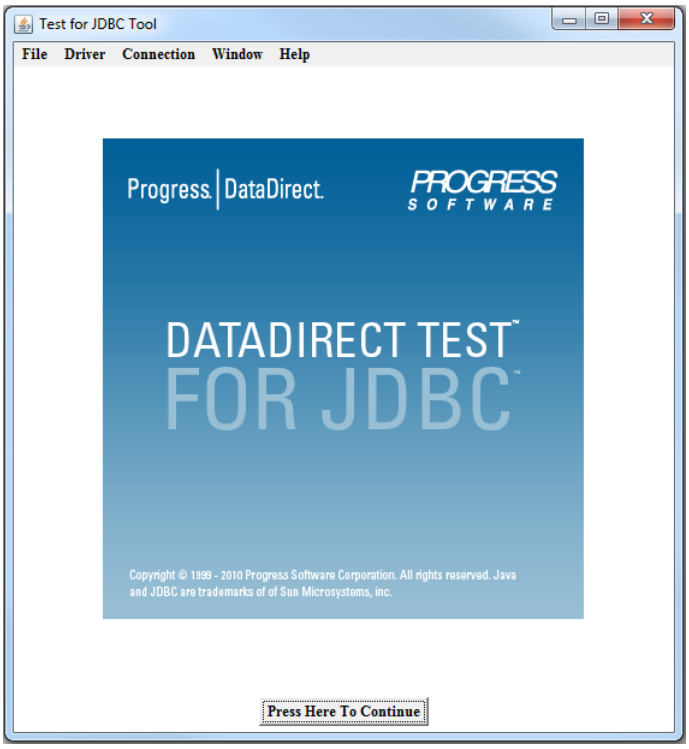
Take the following steps to establish a connection.

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

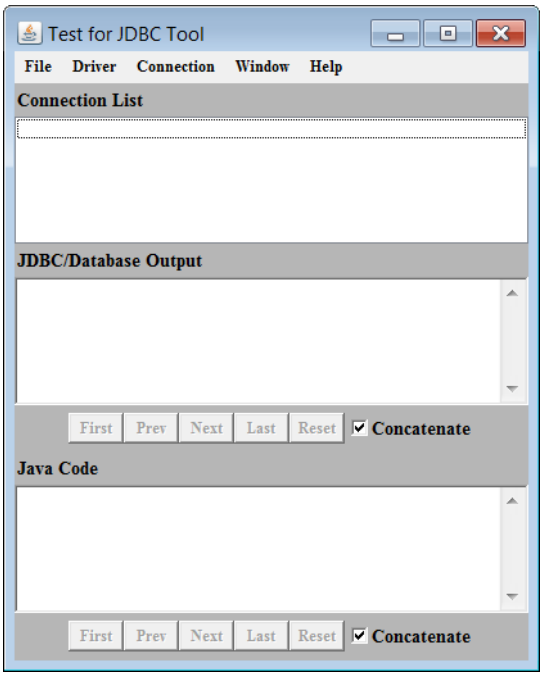
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



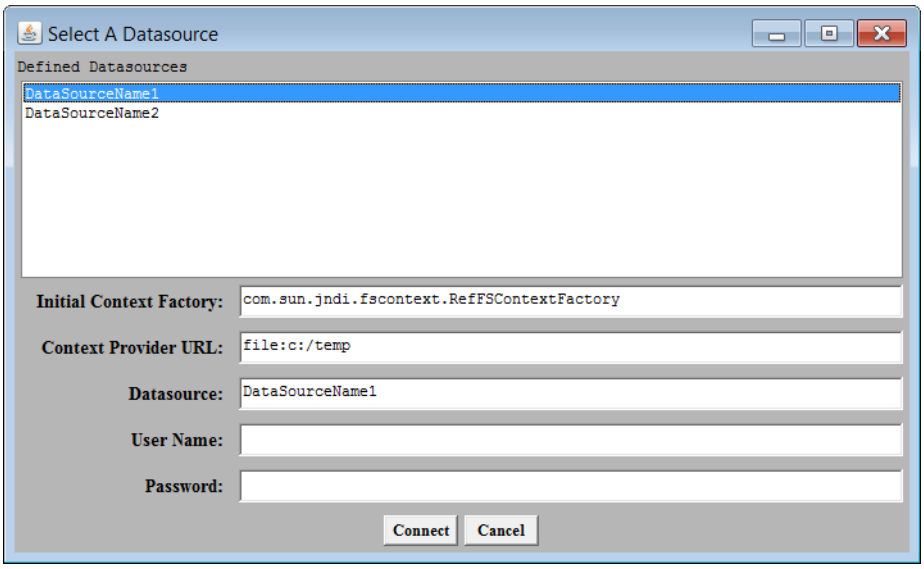
3. Click **Press Here to Continue**.

   The main dialog appears:



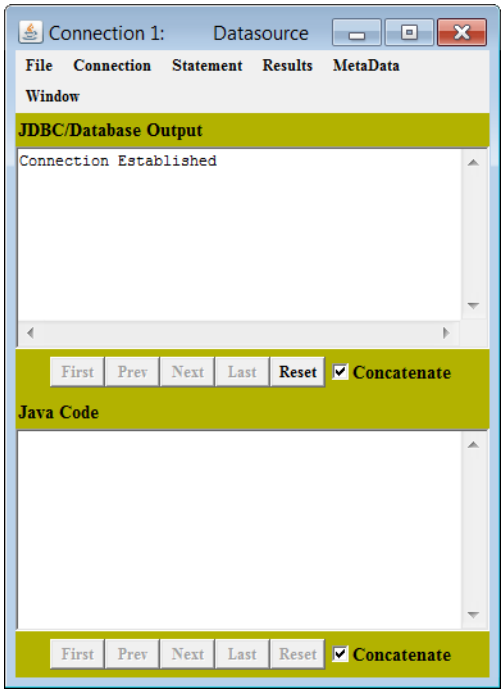
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. If a connection is not established, the window reports an error.



Authentication

The driver supports user ID and password authentication with SCRAM-SHA-256 mechanism. It authenticates the user to the database using a user name and password.

Take the following steps to configure user ID/Password authentication.

1. Specify values for the required properties for establishing a connection.
   a) Set the ServerName property to the IP address or server name of the primary database server.
   b) Set the PortName property to the TCP port of the primary database server that is listening for connections to the database.
2. Set the User property to provide the user name that is used to connect to the database.
3. Set the Password property to specify the password.

For example, the following is a connection string with only the required properties for making a connection using user ID and password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:saphana://MyServer:30015;User=JSmith;Password=secret);
```

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

See also

Password on page 74
User on page 84

Data Encryption

TLS/SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. TLS/SSL negotiates the terms of the encryption in a sequence of events known as the *handshake*. The handshake involves the following types of authentication:

- *TLS/SSL server authentication* requires the server to authenticate itself to the client.
- *TLS/SSL client authentication* is optional and requires the client to authenticate itself to the server after the server has authenticated itself to the client.

Configuring TLS/SSL Encryption

The driver supports TLS/SSL encryption for all supported SAP HANA databases.

Note: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the LoginTimeout property to avoid problems when connecting to a server that does not support SSL.

To configure SSL encryption:

Important: The driver complies with FIPS when FIPS mode is enabled with the client JVM. See "FIPS (Federal Information Processing Standard)" for more information.

- Set the ServerName property to the name or the IP address of the SAP HANA server to which you want to connect. For example, `myserver`.
- Set the PortNumber property to specify the port number of the server listener. The default is `27017`.
- Set the EncryptionMethod property to `SSL`.
- (Optional) Set the CryptoProtocolVersion property to specify acceptable cryptographic protocol versions (for example, TLSv1.3) supported by your server.
- (Optional) Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStorePassword properties or their corresponding Java system properties (`javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`, respectively).
- (Optional) To validate certificates sent by the database server, set the ValidateServerCertificate property to `true`.
- (Optional) Set the HostNameInCertificate property to a host name to be used to validate the certificate. The HostNameInCertificate property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.
- (Optional) If your database server is configured for SSL client authentication, configure your keystore information:
  - Specify the location and password of the keystore file. Either set the KeyStore and KeyStorePassword properties or their corresponding Java system properties (`javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`, respectively).
  - If any key entry in the keystore file is password-protected, set the KeyPassword property to the key password.

The following examples demonstrate the required properties for a session using TLS/SSL encryption with user ID and password authentication.

For a connection URL:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:saphana://myserver:30015;EncryptionMethod=SSL;
   User=JSmith;Password=secret;");
```

For a data source:

```
SAPHANADataSource mds = new SAPHANADataSource();
mds.setDescription("My SAP HANA Data Source");
mds.setEncryptionMethod ("SSL")
mds.setUser("JSmith");
mds.setPassword("secret");
```

See also

Connection property descriptions on page 49
Configuring TLS/SSL Server Authentication on page 45
Configuring TLS/SSL Client Authentication on page 45

Configuring TLS/SSL Server Authentication

When the client makes a connection request, the server presents its public certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted Certificate Authorities (CAs) that resides in an encrypted file on the client known as a *truststore*. Optionally, the client may check the subject (owner) of the certificate. If the certificate matches a trusted CA in the truststore (and the certificate's subject matches the value that the application expects), an encrypted connection is established between the client and server. If the certificate does not match, the connection fails and the driver throws an exception.

To check the issuer of the certificate against the contents of the truststore, the driver must be able to locate the truststore and unlock the truststore with the appropriate password. You can specify truststore information in either of the following ways:

- Specify values for the Java system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword. For example:

  ```
  java -Djavax.net.ssl.trustStore=C:\Certificates\MyTruststore
       -Djavax.net.ssl.trustStorePassword=MyTruststorePassword
  ```

  This method sets values for all TLS/SSL sockets created in the JVM.

- Specify values for the connection properties TrustStore and TrustStorePassword in the connection URL. For example:

  `TrustStore=C:\Certficates\MyTruststore`

  and

  `TrustStorePassword=MyTruststorePassword`

  Any values specified by the TrustStore and TrustStorePassword properties override values specified by the Java system properties. This allows you to choose which truststore file you want to use for a particular connection.

Alternatively, you can configure the drivers to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. If the driver is configured to trust any certificate sent from the server, the issuer information in the certificate is ignored.

Configuring TLS/SSL Client Authentication

If the server is configured for TLS/SSL client authentication, the server asks the client to verify its identity after the server has proved its identity. Similar to TLS/SSL server authentication, the client sends a public certificate to the server to accept or deny. The client stores its public certificate in an encrypted file known as a *keystore*.

The driver must be able to locate the keystore and unlock the keystore with the appropriate keystore password. Depending on the type of keystore used, the driver also may need to unlock the keystore entry with a password to gain access to the certificate and its private key.

The drivers can use the following types of keystores:

- Java Keystore (JKS) contains a collection of certificates. Each entry is identified by an alias. The value of each entry is a certificate and the certificate's private key. Each keystore entry can have the same password as the keystore password or a different password. If a keystore entry has a password different than the keystore password, the driver must provide this password to unlock the entry and gain access to the certificate and its private key.
- PKCS #12 keystores. To gain access to the certificate and its private key, the driver must provide the keystore password. The file extension of the keystore must be .pfx or .p12.

You can specify this information in either of the following ways:

- Specify values for the Java system properties javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword. For example:

```
java -Djavax.net.ssl.keyStore=C:\Certificates\MyKeystore
     -Djavax.net.ssl.keyStorePassword=MyKeystorePassword
```

  This method sets values for all TLS/SSL sockets created in the JVM.

Note: If the keystore specified by the javax.net.ssl.keyStore Java system property is a JKS and the keystore entry has a password different than the keystore password, the KeyPassword connection property must specify the password of the keystore entry (for example, `KeyPassword=MyKeyPassword`).

- Specify values for the connection properties KeyStore and KeyStorePassword in the connection URL. For example:

  `KeyStore=C:\Certficates\MyKeyStore`

  and

  `KeyStorePassword=MyKeystorePassword`

Note: If the keystore specified by the KeyStore connection property is a JKS and the keystore entry has a password different than the keystore password, the KeyPassword connection property must specify the password of the keystore entry (for example, `KeyPassword=MyKeyPassword`).

Any values specified by the KeyStore and KeyStorePassword properties override values specified by the Java system properties. This allows you to choose which keystore file you want to use for a particular connection.

FIPS (Federal Information Processing Standard)

The Federal Information Processing Standard (or FIPS) is a cryptography standard created by the U.S. government. FIPS specifications require certain secure algorithms, cryptographic modules, and random number generation. The driver is FIPS compliant for data encryption when FIPS is enabled for the JVM on the client machine.

The following applies when the driver is running in a FIPS environment:

- The driver complies with 140-3 and 140-2 standards.
- The driver uses PKCS #11 providers to access keystores.

The driver was tested with FIPS 140-3 enabled using Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance.

Performance considerations

EncryptionMethod: Data encryption may adversely affect performance because of the additional overhead (mainly CPU usage) required to encrypt and decrypt data.

FetchSize: FetchSize can be used to adjust the trade-off between throughput and response time. In general, setting larger values for FetchSize will improve throughput, but can reduce response time. You should set FetchSize to suit your environment. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes improve overall fetch times at the cost of additional memory.

InsensitiveResultSetBufferSize: To improve performance when using scroll-insensitive result sets, the driver can cache the result set data in memory instead of writing it to disk. By default, the driver caches 2 MB of insensitive result set data in memory and writes any remaining result set data to disk. Performance can be improved by increasing the amount of memory used by the driver before writing data to disk or by forcing the driver to never write insensitive result set data to disk. The maximum cache size setting is 2 GB.

MaxPooledStatements: To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that does not provide its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created by an application. For example, if the MaxPooledStatements property is set to `20`, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

See also

Connection property descriptions on page 49

4

Connection property descriptions

You can use connection properties to customize the driver for your environment. This section organizes connection properties according to functionality. You can use connection properties with either the JDBC `DriverManager` or a JDBC data source. For a `DriverManager` connection, a property is expressed as a key value pair and takes the form *`property=value`*. For a data source connection, a property is expressed as a JDBC method and takes the form `set`*`property`*`(`*`value`*`)`.

Note:

- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: `{`*`value`*`}`. For example: `User={hello }` or `Password={{hello}}`.

The following tables describe the connection properties by functionality.

- Statement pooling properties
- Failover properties
- Client information properties
- Additional properties

General properties

The following table summarizes the properties used for connecting.

Property	Data Source Method	Default
AuthenticationMethod on page 57	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`userIdPassword`
Password on page 74	`getPassword()` `setPassword(String)`	No default value
PortNumber on page 74	`getPortNumber()` `setPortNumber(int)`	`30015`
ServerName on page 80	`getServerName()` `setServerName(String)`	No default value
User on page 84	`getUser()` `setUser(String)`	No default value

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 75	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 76	`getProxyPassword()` `setProxyPassword(String)`	No default value
ProxyPort on page 76	`getProxyPort()` `setProxyPort(Integer)`	`0` which means the default is determined by the ProxyHost property. For HTTP URLs: `80` For HTTPS URLs: `443`

Property	Data Source Method	Default
ProxyUser on page 77	`getProxyUser()` `setProxyUser(String)`	No default value
UseSystemProxyOptions on page 85	`getUseSystemProxy()` `setUseSystemProxy(Boolean)`	`true`

Data type properties

The following table summarizes connection properties which can be used to handle data types.

Table 3: Data Type Properties

Property	Data Source Method	Default
ConvertNull on page 60	`getConvertNull()` `setConvertNull(Boolean)`	`true`
JavaDoubleToString on page 69	`getJavaDoubleToString()` `setJavaDoubleToString(Boolean)`	`false`

Data encryption properties

The following table summarizes connection properties that can be used to enable TLS/SSL encryption.

Property	Data Source Method	Default
EncryptionMethod on page 62	`setEncryptionMethod` `getEncryptionMethod`	`noEncryption`
HostNameInCertificate on page 66	`setHostNameInCertificate` `getHostNameInCertificate`	No default value
KeyStore on page 70	`setKeyStore` `getKeyStore`	No default value
KeyStorePassword on page 71	`setKeyStorePassword` `getKeyStorePassword`	No default value
KeyPassword on page 69	`setKeyPassword` `getKeyPassword`	No default value
TrustStore on page 83	`setTrustStore` `getTrustStore`	No default value

Property	Data Source Method	Default
TrustStorePassword on page 83	setTrustStorePassword getTrustStorePassword	No default value
ValidateServerCertificate on page 85	setValidateServerCertificate getValidateServerCertificate	true

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
LoginTimeout	getLoginTimeout() setLoginTimeout(Integer)	0
QueryTimeout	getQueryTimeout() setQueryTimeout(Integer)	0

Statement pooling properties

The following table summarizes statement pooling connection properties.

Table 4: Statement Pooling Properties

Property	Data Source Method	Default
ImportStatementPool on page 67	getImportStatementPool() setImportStatementPool(String)	No default value
MaxPooledStatements on page 73	getMaxPooledStatements() setMaxPooledStatements(Integer)	0
RegisterStatementPoolMonitorMBean on page 79	getRegisterStatementPoolMonitorMBean() setRegisterStatementPoolMonitorMBean(Boolean)	false

Failover properties

The following table summarizes connection properties which are used to implement failover.

Property	Data Source Method	Default
AlternateServers on page 55	getAlternateServers() setAlternateServers(String)	No default value

Property	Data Source Method	Default
ConnectionRetryCount on page 59	`getConnectionRetryCount()` `setConnectionRetryCount(Integer)`	`5`
ConnectionRetryDelay on page 60	`getConnectionRetryDelay()` `setConnectionRetryDelay(Integer)`	`1`
DatabaseName on page 61	`getDatabaseName()` `setDatabaseName(String)`	No default value
FailoverGranularity on page 62	`getFailoverGranularity()` `setFailoverGranularity(String)`	`nonAtomic`
FailoverMode on page 63	`getFailoverMode()` `setFailoverMode(String)`	`connect`
FailoverPreconnect on page 64	`getFailoverPreconnect()` `setFailoverPreconnect(Boolean)`	`false`
LoadBalancing on page 71	`getLoadBalancing()` `setLoadBalancing(Boolean)`	No default value

Client information properties

The following table summarizes connection properties which are used to return client information.

Property	Data Source Method	Default
ApplicationName on page 56	`getApplicationName()` `setApplicationName(String)`	No default value
ClientHostName on page 57	`getClientHostName()` `setClientHostName(String)`	No default value
ClientUser on page 58	`getClientUser()` `setClientUser(String)`	No default value
ProgramID on page 78	`getProgramID()` `setProgramID(String)`	No default value

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
FetchSize on page 65	`getFetchSize()` `setFetchSize(Integer)`	`100`
InsensitiveResultSetBufferSize on page 68	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048`
SpyAttributes on page 80	`getSpyAttributes()` `setSpyAttributes(String)`	No default value

For details, see the following topics:

- AlternateServers
- ApplicationName
- AuthenticationMethod
- ClientHostName
- ClientUser
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- DatabaseName
- EncryptionMethod
- FailoverGranularity
- FailoverMode
- FailoverPreconnect
- FetchSize
- HostNameInCertificate
- ImportStatementPool
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- KeyPassword
- KeyStore
- KeyStorePassword
- LoadBalancing
- LoginTimeout

AlternateServers

Purpose

A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See FailoverMode for information about choosing a failover method.

Valid Values

`(servername1[:port1][;property=value[;...]][,servername2[:port2][;property=value[;...]]]...)`

Behavior

The server name (*`servername1`*, *`servername2`*, and so on) is required for each alternate server entry. Port number (*`port1`*, *`port2`*, and so on) and connection properties (*`property=value`*) are optional for each alternate server entry. If the port is unspecified, the port number of the primary server is used. If a port number for the primary server is unspecified, a default port number of 1433 is used. The driver allows only one optional connection property, `DatabaseName`.

Example

The following URL contains alternate server entries for *server2* and *server3*. The alternate server entries contain the optional DatabaseName property.

```
jdbc:datadirect:saphana://server1:1433;DatabaseName=TEST;User=test;Password=secret;
AlternateServers=(server2:1433;DatabaseName=TEST2,server2:1433;DatabaseName=TEST3)
```

Data Source Methods

```
public String getAlternateServers()
```

```
public void setAlternateServers(String)
```

Default Value

No default value

Data Type

String

See also

FailoverMode on page 63

ApplicationName

Purpose

Specifies the name of the application to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

Behavior

The name of the application to be stored in the database. This property sets the program_name column in the sysprocesses table in the database.

Notes

- ProgramName can be used as an alias for ApplicationName.
- Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it.

Data Source Methods

```
public String getApplicationName()
```

```
public void setApplicationName(String)
```

Default Value

No default value

Data Type

String

AuthenticationMethod

Purpose

Determines which authentication mechanism the driver uses when establishing a connection.

Valid Values

`userIdPassword`

Behavior

If set to `userIdPassword`, the driver uses `SCRAM-SHA-256` user ID and password authentication. The user property provides the user ID, and the Password property provides the password.

Notes

- The `userIdPassword` setting is appropriate for servers that do not have authentication enabled. In a scenario where AuthenticationMethod has been set to `userIdPassWord` but the server has not been configured for user ID/password authentication, the driver will still connect to the database server.
- If authentication has not been enabled, client applications will have access to all databases on the server. If authentication has been enabled, a client application will only have access to the database specified by the DatabaseName property assuming it has the required permissions. However, an application with clusterAdmin privileges will have access to all databases on the server even when authentication is enabled.

Data Source Methods

```
public String getAuthenticationMethod()
```

```
public void setAuthenticationMethod(String)
```

Default Value

`userIdPassword`

Data Type

String

ClientHostName

Purpose

Specifies the host name of the client machine to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the host name of the client machine.

Data Source Method

```
public String getClientHostName()
public void setClientHostName(String)
```

Default

No default value

Data Type

String

ClientUser

Purpose

Specifies the user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is a valid user ID.

Data Source Method

```
public String getClientUser()
public void setClientUser(String)
```

Default

No default value

Data Type

String

ConnectionRetryCount

Purpose

The number of times the driver retries connection attempts to the SAP HANA database until a successful connection is established.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents the number of retries.

Behavior

If set to `0`, the driver does not try to reconnect after the initial unsuccessful attempt.

If set to *x*, the driver retries connection attempts the specified number of times. If a connection is not established during the retry attempts, the driver returns an exception that is generated by the last server to which it tried to connect.

Example

If this property is set to `2`, the driver retries the server twice after the initial retry attempt.

Notes

- If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), and the login timeout expires, the driver ceases connection attempts.
- The ConnectionRetryDelay property specifies the wait interval, in seconds, to occur between retry attempts.

Data Source Methods

```
public Integer getConnectionRetryCount()
```

```
public void setConnectionRetryCount(Integer)
```

Default Value

5

Data Type

Integer

ConnectionRetryDelay

Purpose

The number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer.

Valid Values

`0` | *x*

where:

x

is a number of seconds.

Behavior

If set to `0`, the driver does not delay between retries.

If set to *x*, the driver waits between connection retry attempts the specified number of seconds.

Example

If ConnectionRetryCount is set to 2 and this property is set to 3, the driver retries the server twice after the initial retry attempt. The driver waits 3 seconds between retry attempts.

Data Source Methods

```
public Integer getConnectionRetryDelay()
```

```
public void setConnectionRetryDelay(Integer)
```

Default Value

`1`

Data Type

Integer

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of whether the column value is `NULL`.

If set to `false`, the driver does not perform the data type check if the value of the column is `NULL`. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.

Data Source Methods

```
public Boolean getConvertNull()
```

```
public void setConvertNull(Boolean)
```

Default Value

`true`

Data Type

Boolean

DatabaseName

Purpose

Specifies the name of the database to which you are connecting.

Valid Values

`database_name`

where:

`database_name`

is the name of a valid database.

Important: The value is case-insensistive if you have access privileges to query the list of databases on the server. If you do not have access, the value is case-sensitive.

Data Source Methods

```
public String getDatabaseName()
```

```
public void setDatabaseName(String)
```

Default Value

No default value

Data Type

String

EncryptionMethod

Purpose

Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server.

Valid Values

`noEncryption | SSL`

Behavior

If set to `noEncryption`, data is not encrypted or decrypted.

If set to `SSL`, data is encrypted using TLS/SSL. If the database server does not support TLS/SSL, the connection fails and the driver throws an exception.

Data Source Methods

```
public String getEncryptionMethod()
```

```
public void setEncryptionMethod(String)
```

Default

`noEncryption`

Data Type

String

See also

Performance considerations

FailoverGranularity

Purpose

Determines how the driver behaves if exceptions occur while trying to reestablish a lost connection. This property is ignored if `FailoverMode=connect`.

Valid values

`nonAtomic` | `atomic` | `atomicWithRepositioning`

Behavior

If set to `nonAtomic`, the driver continues with the failover process and posts any exceptions on the statement on which they occur.

If set to `atomic`, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. The driver stops trying to connect to an alternative server and returns an exception indicating that the connection was lost. If an exception is generated as a result of restoring the state of work in progress by re-executing the Select statement, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.

If set to `atomicWithRepositioning`, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress. The driver stops trying to connect to an alternative server and returns an exception indicating that the connection was lost.

Data source method

```
public String getFailoverGranularity()
```

```
public void setFailoverGranularity(String)
```

Default

`nonAtomic`

Data type

String

See also

- FailoverMode on page 63

FailoverMode

Purpose

Specifies the type of failover method the driver uses.

Valid Values

`connect` | `extended` | `select`

Behavior

If set to `connect`, the driver provides failover protection for new connections only.

If set to `extended`, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to `select`, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work that is performed by the last Select statement that was executed on the Statement object.

Notes

- The AlternateServers property specifies one or multiple alternate servers for failover and is required for all failover methods. To turn off failover, do not specify a value for the AlternateServers property.
- The FailoverGranularity property determines which action the driver takes if exceptions occur during the failover process.

- The FailoverPreconnect property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Data Source Methods

```
public String getFailoverMode()
```

```
public void setFailoverMode(String)
```

Default

`connect`

Data Type

String

See also

AlternateServers on page 55
FailoverGranularity on page 62
FailoverPreconnect on page 64

FailoverPreconnect

Purpose

Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if `FailoverMode=connect`.

Valid values

`true` | `false`

Behavior

If set to `true`, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to `false`, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

Notes

The AlternateServers property specifies one or multiple alternate servers for failover.

Data source method

```
public Boolean getFailoverPreconnect()
```

```
public void setFailoverPreconnect(Boolean)
```

Default

`false`

Data type

Boolean

See also

- AlternateServers on page 55
- FailoverMode on page 63

FetchSize

Purpose

Specifies the maximum number of rows that the driver processes before returning data to the application when executing a Select. This value provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

Valid Values

`0` | *x*

where:

x

is a positive integer indicating the number of rows that should be processed.

Behavior

If set to `0`, the driver processes all the rows of the result before returning control to the application. When large data sets are being processed, setting FetchSize to `0` can diminish performance and increase the likelihood of out-of-memory errors.

If set to *x*, the driver limits the number of rows that may be processed for each fetch request before returning control to the application.

Notes

- To optimize throughput and conserve memory, the driver uses an internal algorithm to determine how many rows should be processed based on the width of rows in the result set. Therefore, the driver may process fewer rows than specified by FetchSize when the result set contains exceptionally wide rows. Alternatively, the driver processes the number of rows specified by FetchSize when the result set contains rows of unexceptional width.
- FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.
- You can use FetchSize to reduce demands on memory and decrease the likelihood of out-of-memory errors. Simply, decrease FetchSize to reduce the number of rows the driver is required to process before returning data to the application.

Data Source Methods

```
public Integer getFetchSize()
```

```
public void setFetchSize(Integer)
```

Default Value

100

Data Type

Integer

See also

Performance considerations on page 47

HostNameInCertificate

Purpose

Specifies a host name for certificate validation when TLS/SSL encryption is enabled (EncryptionMethod=SSL) and validation is enabled (ValidateServerCertificate=true). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

Valid Values

host_name | #SERVERNAME#

where:

host_name

is a valid host name.

Behavior

If *host_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate. If the values do not match, the connection fails and the driver throws an exception.

If #SERVERNAME# is specified, the driver compares the server name that is specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If the certificate does not have a SubjectAlternativeName, the driver compares the host name to the CN part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If multiple CN parts are present, the driver validates the host name against each CN part. If any one validation succeeds, a connection is established.

Notes

- If TLS/SSL encryption or certificate validation is not enabled, this property is ignored.
- If TLS/SSL encryption and validation is enabled and this property is unspecified, the driver uses the server name specified in the connection URL or data source of the connection to validate the certificate.

Data Source Methods

```
public String getHostNameInCertificate()
```

```
public void setHostNameInCertificate(String)
```

Default

No default value

Data Type

String

ImportStatementPool

Purpose

Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

Valid Values

String

where:

String

is the path and file name of the file to be used to load the contents of the statement pool.

Notes

- If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getImportStatementPool()
```

```
public void setImportStatementPool(String)
```

Default Value

No default value

Data Type

String

InsensitiveResultSetBufferSize

Purpose

Determines the amount of memory that is used by the driver to cache insensitive result set data.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents the amount of memory.

Behavior

If set to `-1`, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. With no need to write result set data to disk, the driver processes the data efficiently.

If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 MB. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk.

If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in efficient memory use.

Data Source Methods

```
public Integer getInsensitiveResultsetBufferSize()
```

```
public void setInsensitiveResultsetBufferSize(Integer)
```

Default Value

`2048`

Data Type

Integer

See also

Performance considerations on page 47

JavaDoubleToString

Purpose

Determines which algorithm the driver uses when converting a double or float value to a string value. By default, the driver uses its own internal conversion algorithm, which improves performance.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses the JVM algorithm when converting a double or float value to a string value. If your application cannot accept rounding differences and you are willing to sacrifice performance, set this value to `true` to use the JVM conversion algorithm.

If set to `false`, the driver uses its own internal algorithm when converting a double or float value to a string value. This value improves performance, but slight rounding differences within the allowable error of the double and float data types can occur when compared to the same conversion using the JVM algorithm.

Data Source Method

```
public Boolean getJavaDoubleToString()
```

```
public void setJavaDoubleToString(Boolean)
```

Default

`false`

Data Type

boolean

KeyPassword

Purpose

Specifies the password that is used to access the individual keys in the keystore file when TLS/SSL is enabled (`EncryptionMethod=SSL`) and TLS/SSL client authentication is enabled on the database server. This property is useful when individual keys in the keystore file have a different password than the keystore file.

Valid Values

`string`

where:

`string`

is a valid password.

Data Source Methods

```
public String getKeyPassword()
```

```
public void setKeyPassword(String)
```

Default

No default value

Data Type

String

KeyStore

Purpose

Specifies the directory of the keystore file to be used when TLS/SSL is enabled (EncryptionMethod=SSL) and TLS/SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.

This value overrides the directory of the keystore file that is specified by the javax.net.ssl.keyStore Java system property. If this property is not specified, the keystore directory is specified by the javax.net.ssl.keyStore Java system property.

Valid Values

string

where:

string

is a valid directory of a keystore file.

Notes

- The keystore and truststore files can be the same file.

Data Source Methods

```
public String getKeyStore()
```

```
public void setKeyStore(String)
```

Default

No default value

Data Type

String

KeyStorePassword

Purpose

Specifies the password that is used to access the keystore file when TLS/SSL is enabled (EncryptionMethod=SSL) and TLS/SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.

This value overrides the password of the keystore file that is specified by the javax.net.ssl.keyStorePassword Java system property. If this property is not specified, the keystore password is specified by the javax.net.ssl.keyStorePassword Java system property.

Notes

- The keystore and truststore files can be the same file.

Valid Values

string

where:

string

is a valid password.

Data Source Methods

```
public String getKeyStorePassword()
```

```
public void setKeyStorePassword(String)
```

Default

No default value

Data Type

String

LoadBalancing

Purpose

Determines whether the driver uses client load balancing in its attempts to connect to the database servers (primary and alternate). You can specify one or multiple alternate servers by setting the AlternateServers property.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses client load balancing and attempts to connect to the database servers (primary and alternate) in random order. The driver randomly selects from the list of primary and alternate servers which server to connect to first. If that connection fails, the driver again randomly selects from this list of servers until all servers in the list have been tried or a connection is successfully established.

If set to `false`, the driver does not use client load balancing and connects to each server based on their sequential order (primary server first, then, alternate servers in the order they are specified).

Data Source Methods

```
public Boolean getLoadBalancing()
```

```
public void setLoadBalancing(Boolean)
```

Default Value

`false`

Data Type

Boolean

See also

AlternateServers on page 55

LoginTimeout

Purpose

The amount of time, in seconds, that the driver waits for a connection to be established before timing out the connection request.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to `-1` | `0`, the driver does not time out a connection request.

If set to *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.

Data Source Methods

```
public Integer getLoginTimeout()
```

```
public void setLoginTimeout(Integer)
```

Default Value

0

Data Type

Integer

MaxPooledStatements

Purpose

Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling.

Valid Values

0 | *x*

where:

x

is a positive integer that represents a number of prepared statements to be cached.

Behavior

If set to 0, the driver's internal prepared statement pooling is not enabled.

If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Example

If the value of this property is set to 20, the driver caches the last 20 prepared statements that are created by the application.

Notes

When you enable statement pooling, your applications can access the Statement Pool Monitor directly with DataDirect-specific methods. However, you can also enable the Statement Pool Monitor as a JMX MBean. To enable the Statement Pool Monitor as an MBean, statement pooling must be enabled with MaxPooledStatements and the Statement Pool Monitor MBean must be registered using the RegisterStatementPoolMonitorMBean connection property.

Data Source Methods

```
public Integer getMaxPooledStatements()
```

```
public void setMaxPooledStatements(Integer)
```

Default Value

`0`

Data Type

Integer

See also

Performance considerations on page 47

Password

Purpose

A password that is used to connect to the service.

Important: Setting the password using a data source is not recommended. The data source persists all properties, including password, in clear text.

Behavior

password

where:

password

is a valid password. The password is case-sensitive.

Data Source Methods

```
public String getPassword()
```

```
public void setPassword(String)
```

Default Value

No default value

Data Type

String

See also

User ID/password authentication on page 11

PortNumber

Purpose

Specifies the port number of the server listener.

Valid Values

port_number

where:

port_number

is the port number of the server listener. Check with your database administrator for the correct number.

Data Source Methods

```
public Integer getPortNumber()
public void setPortNumber(Integer)
```

Default Value

30015

Data Type

Integer

ProxyHost

Purpose

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

is the name of the proxy server, which may be qualified with the domain name.

IP_address

is an IP address, specified in either IPv4 or IPv6 format, or a combination of the two.

Data Source Methods

```
public String getProxyHost()
public void setProxyHost(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 12

ProxyPassword

Purpose

Specifies the password needed to connect to a proxy server for the first connection.

Valid Values

password

where:

password

is a valid password for that server. Contact your system administrator to obtain a valid password.

Data Source Methods

```
public String getProxyPassword()
```

```
public void setProxyPassword(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 12

ProxyPort

Purpose

Specifies the port number where the proxy server is listening for HTTP or HTTPS requests for the first connection.

Valid Values

port

where:

port

is the port number on which the proxy server is listening. Contact your system administrator to obtain the correct port.

Data Source Methods

```
public Integer getProxyPort()
```

```
public void setProxyPort(Integer)
```

Default Value

`0` which means that the default value is determined by whether the value specified for the ProxyHost property is an HTTP or HTTPS URL.

For HTTP: `80`

For HTTPS: `443`

Data Type

Integer

See also

Proxy server on page 12

ProxyUser

Purpose

Specifies the user name needed to connect to a proxy server for the first connection.

Valid Values

`user_name`

where:

`user_name`

is a valid user ID for the proxy server.

Data Source Methods

```
public String getProxyUser()
```

```
public void setProxyUser(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server on page 12

ProgramID

Purpose

The driver and version information on the client to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

`string`

where:

`string`

is a value that identifies the product and version of the driver on the client.

Example

`DDJ04200`

Data Source Methods

```
public String getProgramID()
```

```
public void setProgramID(String)
```

Default

No default value

Data Type

String

QueryTimeout

Purpose

Sets the default query timeout (in seconds) for all statements created by a connection.

Valid Values

`-1` | `0` | *`x`*

where:

`x`

is a number of seconds.

Behavior

If set to `-1`, the query timeout functionality is disabled. The driver silently ignores calls to the Statement.setQueryTimeout() method.

If set to `0`, the default query timeout is infinite (the query does not time out).

If set to *x*, the driver uses the value as the default timeout for any statement that is created by the connection. To override the default timeout value that is set by this property, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.

Data Source Methods

```
public Integer getQueryTimeout()
```

```
public void setQueryTimeout(Integer)
```

Default Value

`0`

Data Type

Integer

RegisterStatementPoolMonitorMBean

Purpose

Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled.

If set to `false`, the driver does not register an MBean for the Statement Pool Monitor for any statement pool.

Notes

- Registering the MBean exports a reference to the Statement Pool Monitor. The exported reference can prevent garbage collection on connections if the connections are not properly closed. When garbage collection does not take place on these connections, out of memory errors can occur.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMbean()
```

```
public void setRegisterStatementPoolMonitorMbean(Boolean)
```

Default Value

`false`

Data Type

Boolean

ServerName

Purpose

Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.

Valid Values

`string`

where:

`string`

is a valid IP address or server name.

Data Source Methods

```
public String getServerName()
```

```
public void setServerName(String)
```

Default Value

No default value

Data Type

String

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

Valid Values

(*`spy_attribute`*[;*`spy_attribute`*]...)

where:

spy_attribute

is any valid DataDirect spy attribute.

Behavior

Attribute	Description
linelimit=*numberofchars*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
load=*classname*	Loads the driver specified by *classname*.
log=(file)*filename*	Directs logging to the file specified by *filename*. For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes.
log=(filePrefix)*file_prefix*	Directs logging to a file prefixed by *file_prefix*. The log file is named *file_prefixX*.log where: *X* is an integer that increments by 1 for each connection on which the prefix is specified. For example, if the attribute log=(filePrefix)C:\\temp\\spy_ is specified on multiple connections, the following logs are created: C:\temp\spy_1.log C:\temp\spy_2.log C:\temp\spy_3.log ... If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes.
log=System.out	Directs logging to the Java output standard, System.out.

Attribute	Description
logIS={ yes | no | nosingleread }	Specifies whether DataDirect Spy logs activity on InputStream and Reader objects. When logIS=nosingleread, logging on InputStream and Reader objects is active; however, logging of the single-byte read InputStream.read or single-character Reader.read is suppressed to prevent generating large log files that contain single-byte or single character read messages. The default is no.
logLobs={ yes | no }	Specifies whether DataDirect Spy logs activity on BLOB and CLOB objects.
logTName={ yes | no }	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp={ yes | no }	Specifies whether a timestamp is included on each line of the DataDirect Spy log. The default is no.

Example

The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.

```
(log=(file)/tmp/spy.log;linelimit=80)
```

Notes

- If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\spy.log.
- For more information, refer to "Tracking JDBC calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getSpyAttributes()
```

```
public void setSpyAttributes(String)
```

Default Value

No default value

Data Type

String

TrustStore

Purpose

Specifies the directory of the truststore file to be used when TLS/SSL is enabled (EncryptionMethod=SSL) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.

This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property.

This property is ignored if ValidateServerCertificate=`false`.

Valid Values

`string`

The directory of the truststore file.

Data Source Methods

```
public String getTrustStore()
```

```
public void setTrustStore(String)
```

Default

No default value

Data Type

String

TrustStorePassword

Purpose

Specifies the password that is used to access the truststore file when TLS/SSL is enabled (EncryptionMethod=SSL) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.

This value overrides the password of the truststore file that is specified by the javax.net.ssl.trustStorePassword Java system property. If this property is not specified, the truststore password is specified by the javax.net.ssl.trustStorePassword Java system property.

This property is ignored if ValidateServerCertificate=`false`.

Valid Values

string

where:

string

is a valid password for the truststore file.

Data Source Methods

```
public String getTrustStorePassword()
public void setTrustStorePassword(String)
```

Default

No default value

Data Type

String

User

Purpose

Specifies the user name that is used to connect to the service.

Valid Values

String

where:

String

is a valid user name. The user name is case-insensitive.

Data Source Methods

```
public String getUser()
public void setUser(String)
```

Default Value

No default value

Data Type

String

See also

User ID/password authentication on page 11

UseSystemProxyOptions

Purpose

Determines whether the driver attempts to use JVM system properties to configure proxy server settings by default.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver attempts to use the settings of the `http.proxyHost` and `http.proxyPort` JVM system properties. If no proxy server settings are configured on the JVM, the driver uses the properties specified.

If set to `false`, the driver uses only proxy server properties configured in the connection string or datasource. Specify this value when the driver is in a JVM environment with other Java applications, but you do not want to connect through the proxy server that is JVM system wide.

Data Source Methods

```
public Boolean getUseSystemProxy()
```

```
public void setUseSystemProxy(Boolean)
```

Default Value

`true`

Data Type

Boolean

See also

Proxy server on page 12

ValidateServerCertificate

Purpose

Determines whether the driver validates the certificate that is sent by the database server when TLS/SSL encryption is enabled (EncryptionMethod=SSL). When using TLS/SSL server authentication, any certificate that is sent by the server must be issued by a trusted Certificate Authority (CA). Allowing the driver to trust any certificate that is returned from the server even if the issuer is not a trusted CA is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver validates the certificate that is sent by the database server. Any certificate from the server must be issued by a trusted CA in the truststore file. If the HostNameInCertificate property is specified, the driver also validates the certificate using a host name. The HostNameInCertificate property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

If set to `false`, the driver does not validate the certificate that is sent by the database server. The driver ignores any truststore information that is specified by the TrustStore and TrustStorePassword properties or Java system properties.

Notes

- Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties.

Data Source Methods

```
public Boolean getValidateServerCertificate()
```

```
public void setValidateServerCertificate(Boolean)
```

Default

`true`

Data Type

Boolean