Progress DataDirect for JDBC for Db2

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Db2 Driver

The Progress® DataDirect® for JDBC™ for Db2™ driver (Db2 driver) supports the JDBC API for SQL read-write access to:

- Db2 for i
- Db2 for Linux, UNIX, and Windows
- Db2 for z/OS
- Db2 Hosted
- Db2 Warehouse on Cloud

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-ibm-db2.

For details, see the following topics:

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 GA

- **Enhancements**
  - 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 68 for details.
- **Changed Behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See SpyAttributes on page 161 for details.

Changes for 6.0.0 GA

- **Driver Enhancements**
  - The driver has been enhanced to support the TLSv1.3 cryptographic protocol. As part of this enhancement, the default cryptographic protocol enabled by the driver has been updated to TLSv1.3. See CryptoProtocolVersion on page 123 for more information.
  - The driver has been enhanced to support the Generic Security Standard (GSS) plug-in for authentication. The plug-in consists of a set of APIs that you can implement to customize your authentication requirements and interoperate with various security methods. You can configure a GSS plug-in using the refreshed AuthenticationMethod property and the new GSSPluginName and GSSPluginObject properties. For details, see GSS plug-in authentication on page 62.
  - The driver has been enhanced to support connecting to a proxy server through an HTTP connection. HTTP proxy support is configurable with the new ProxyHost, ProxyMode, ProxyPassword, ProxyPort, and ProxyUser connection properties. For details, see Proxy server on page 69.

- The KeepAlive connection property has been added to the driver. Enabling the KeepAlive property allows the client to keep idle TCP connections active by periodically passing packets to the server. For details, see KeepAlive on page 138.
- The driver has been enhanced to support Windows Defender Credential Guard when using Kerberos Authentication. For details, see Kerberos authentication on page 56
- The driver has been enhanced to include timestamp in the Spy and JDBC packet logs by default. If required, you can disable the timestamp logging by specifying the following at connection: For Spy logs, set `spyAttributes=(log=(file)Spy.log;timestamp=no)` and for JDBC packet logs, set `ddtdbg.ProtocolTraceShowTime=false`.
- 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 37 for details.
- The RegisterStatementPoolMonitorMBean connection property has been added. Note that the driver no longer registers the Statement Pool Monitor as a JMX MBean by default. You must set RegisterStatementPoolMonitorMBean to `true` to register the Statement Pool Monitor and manage statement pooling with standard JMX API calls. See RegisterStatementPoolMonitorMBean on page 156 for details.
- The driver has been enhanced to support cursor type OUT parameters for Db2 for Linux, UNIX, Windows stored procedures. For details, see Db2 for Linux, UNIX, Windows stored procedure cursor type OUT parameters on page 88.

- **Changed Behavior**
  - The TLSv1.1 and TLSv1.0 cryptographic protocols are now disabled by default. These protocols are no longer considered secure and, therefore, are no longer recommended for use. However, the driver still supports TLSv1.1 and TLSv1.0 for legacy servers that do not support more secure protocols. See CryptoProtocolVersion on page 123 for more information.
  - The SSLv3 and SSLv2 cryptographic protocols are no longer supported.
  - For Kerberos authentication, the driver no longer sets the java.security.krb5.conf system property to force the use of the krb5.conf file installed with the driver jar files in the /lib directory of the product installation directory. For details, see Kerberos authentication on page 56.
  - For Db2 for z/OS, the AlternateID connection property has been modified to set the name of the schema in the DB2 CURRENT SCHEMA special register instead of the DB2 CURRENT SQLID special register. AlternateID now sets the name of the schema in the CURRENT SCHEMA special register for Db2 for i, Db2 for Linux/UNIX/Windows, and Db2 for z/OS. For details, see AlternateID on page 108.

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\db2.jar
     ```

   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/db2.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 examples show how to connect using user ID/password authentication.

     - **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

       ```
       jdbc:datadirect:db2://myserver1:50000;
       DatabaseName=account;User=jsmith;Password=secret;
       ```

     - **For Db2 for i and Db2 for z/OS**:

       ```
       jdbc:datadirect:db2://myserver2:446;
       LocationName=Sample;User=test;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.

Note: See Authentication on page 53 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:
   - Interactive SQL for JDBC (JDBCISQL): JDBCISQL 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.
   - 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.
   - 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.db2.DB2Driver

DataSource class:

com.ddtek.jdbcx.db2.DB2DataSource

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:

- 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 and password authentication

This string includes the properties used to connect using user ID and password authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;
DatabaseName=MyDB;User=jsmith;Password=secret;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:446;
LocationName=MyLocation;User=test;Password=secret;
```

For more information on these properties and values, see User ID/password authentication on page 54.

User ID and password authentication with AES encryption

This string includes the properties used to connect using user ID and password authentication and encrypt data using AES encryption with random number generator for secure seeding.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;DatabaseName=MyDB;
AuthenticationMethod=encryptedUIDPasswordAES;RandomGenerator=random;
SecureRandomAlgorithm=SHA1PRNG;User=jsmith;Password=secret;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver2:446;LocationName=MyLocation;
AuthenticationMethod=encryptedUIDPasswordAES;RandomGenerator=random;
SecureRandomAlgorithm=SHA1PRNG;User=jsmith;Password=secret;
```

For more information on these properties and values, see Random number generator secure seeding on page 55.

Client authentication

This string includes the properties used to connect using client authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;
DatabaseName=MyDB;AuthenticationMethod=client;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:446;
LocationName=MyLocation;AuthenticationMethod=client;
```

For more information on these properties and values, see Client authentication on page 61.

Kerberos authentication

This string includes the properties used to connect using Kerberos authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;
DatabaseName=MyDB;AuthenticationMethod=kerberos;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:446;
LocationName=MyLocation;AuthenticationMethod=kerberos;
```

This authentication method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos and MIT Kerberos. For more information, see Kerberos authentication on page 56.

GSS plug-in authentication

This string includes the properties used to connect using GSS plug-in authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;
DatabaseName=MyDB;AuthenticationMethod=pluginSecurity;
GSSPluginName=gssapi_name;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:446;
LocationName=MyLocation;AuthenticationMethod=pluginSecurity;
GSSPluginName=gssapi_name;
```

In addition to the connection properties mentioned in the above examples, you need to specify a value for the GSSPluginObject connection property using the Java properties in the client application, as it cannot be passed as part of the connection URL. For example:

```
java.util.Properties properties = new java.util.Properties();
properties.put("GSSPluginObject", new DB2GSSPlugin());
Connection connection= DriverManager.getConnection(connectionUrl, properties);
```

For more information on these properties and values, see GSS plug-in authentication on page 62.

Connection failover

This string includes the properties used to connect using connection failover.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver1:50000;DatabaseName=MyDB;
User=jsmith;Password=secret;AlternateServers=(myserver2:50000;
myserver3:50000);ConnectionRetryCount=2;ConnectionRetryDelay=5;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver1:446;LocationName=MyLocation;
User=jsmith;Password=secret;AlternateServers=(myserver2:446;
myserver3:446);ConnectionRetryCount=2;ConnectionRetryDelay=5;
```

For more information on these properties and values, see Failover on page 70.

Proxy server

This string includes the properties used to connect using connection failover.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;DatabaseName=MyDB;
ProxyHost=pserver;ProxyPort=1234;ProxyUser=jsmith;
ProxyPassword=proxys3cr3t;User=jsmith@abc.com;Password=secret;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:446;LocationName=MyLocation;
ProxyHost=pserver;ProxyPort=1234;ProxyUser=jsmith;
ProxyPassword=proxys3cr3t;User=jsmith@abc.com;Password=secret;
```

For more information on these properties and values, see Proxy server on page 69.

TLS/SSL server encryption

This string includes the properties used for TLS/SSL server encryption with user ID/password authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;DatabaseName=MyDB;
EncryptionMethod=ssl;TrustStore=TrustStorePath;
TrustStorePassword=secrettruststore;User=jsmith;
Password=secret;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:50000;LocationName=MyLocation;
EncryptionMethod=ssl;TrustStore=TrustStorePath;
TrustStorePassword=secrettruststore;User=jsmith;
Password=secret;
```

For more information on these properties and values, see Configuring TLS/SSL encryption on page 65.

TLS/SSL client encryption

This string includes the properties used for TLS/SSL client encryption with user ID/password authentication.

- **For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud**:

```
jdbc:datadirect:db2://myserver:50000;DatabaseName=MyDB;
EncryptionMethod=ssl;KeyStore=KeyStorePath;
KeyStorePassword=secretkeystore;User=jsmith;
Password=secret;
```

- **For Db2 for i and Db2 for z/OS**:

```
jdbc:datadirect:db2://myserver:50000;LocationName=MyLocation;
EncryptionMethod=ssl;KeyStore=KeyStorePath;
KeyStorePassword=secretkeystore;User=jsmith;
Password=secret;
```

For more information on these properties and values, see Configuring TLS/SSL encryption on page 65.

Data types

The following table lists the data types supported by the Db2 driver and describes how they are mapped to JDBC data types.

Table 1: DB2 Data Types

DB2 Data Type	JDBC Data Type
Bigint	BIGINT
Binary[1]	BINARY
Blob	BLOB
Char	CHAR
Char for Bit Data	BINARY
Clob	CLOB
Date	DATE or TIMESTAMP[2]

[1] Supported only for Db2 for z/OS.

[2] For Db2 for Linux/UNIX/Windows with the Oracle compatibility feature enabled, the Date type maps to the JDBC TIMESTAMP type.

DB2 Data Type	JDBC Data Type
DBClob	CLOB or NCLOB[3]
Decfloat	DECIMAL
Decimal	DECIMAL
Double	DOUBLE
Float	DOUBLE
Graphic	CHAR or NCHAR[3]
Integer	INTEGER
Long Varchar	LONGVARCHAR
Long Varchar for Bit Data	LONGVARBINARY
Long Vargraphic	LONGVARCHAR or LONGNVARCHAR[3]
Numeric	NUMERIC
Real	REAL
Rowid[4]	VARBINARY
Smallint	SMALLINT
Time	TIME
Timestamp	TIMESTAMP
Timestamp with Time Zone[5]	VARCHAR or TIMESTAMP[6]
Varbinary	VARBINARY
Varchar	VARCHAR
Varchar for Bit Data	VARBINARY
Vargraphic	VARCHAR or NVARCHAR[3]
XML[7,8]	CLOB or SQLXML[3]

[3] When JDBCBehavior=`1`, the first value applies and when JDBCBehavior=`0`, the second value applies.
[4] Supported only for Db2 for z/OS and Db2 for i
[6] When FetchTSWTZasTimestamp=`false` (default), this data type is mapped to the JDBC VARCHAR data type; when FetchTSWTZasTimestamp=`true`, it is mapped to the JDBC TIMESTAMP data type.
[5] Supported only for Db2 for z/OS.
[7] Supported only for Db2 for Linux/UNIX/Windows and Db2 for i V7R1 and higher.
[8] The XML DescribeType property overrides the mappings for XML data.

See "Large object (LOB) support" for more information about the Blob, Clob, and DBClob data types. See "Returning and inserting/updating XML data" for more information about the XML data type.

See also

Large object (LOB) support on page 88

getTypeInfo()

The following table provides getTypeInfo() results for all Db2 databases supported by the Db2 driver.

Table 2: getTypeInfo() for Db2

TYPE_NAME = bigint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bigint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = binary [9] AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = BINARY(X' LITERAL_SUFFIX = ') LOCAL_TYPE_NAME = binary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[9] Supported only for Db2 for z/OS.

TYPE_NAME = blob AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = (*length*) DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BLOB MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = (*length*) DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 (DB2 for Linux/UNIX/Windows), 255 (DB2 for z/OS), 32765 (DB2 for i) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char() for bit data AUTO_INCREMENT = NULL NULL CASE_SENSITIVE = false CREATE_PARAMS = (*length*) DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char() for bit data MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 (Db2 for Linux/UNIX/Windows), 254 (Db2 for z/OS), 32765 (Db2 for i) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = clob AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = (*length*) DATA_TYPE = 2005 (CLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = clob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = date MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = dbclob AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = (*length*) (Db2 for Linux/UNIX/Windows and Db2 for z/OS), (*length*) CCSID 13488 (Db2 for i) DATA_TYPE = 2005 (CLOB) [10] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = dbclob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[10] If JDBCBehavior=0, the value returned for DATA_TYPE is 2011 (NCLOB). If JDBCBehavior=1, the value returned is 2005 (CLOB).

TYPE_NAME = decfloat AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *precision* DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NULL MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 53 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = decimal AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = (*precision*,*scale*) DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal MAXIMUM_SCALE = 31	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 31 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = double AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = double MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 53 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

TYPE_NAME = float AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = graphic AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = 1 (CHAR) [11] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 127 (DB2 for Linux/UNIX/Windows), 127 (Db2 for z/OS), 16383 (Db2 for i) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = integer AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = integer MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

[11] If JDBCBehavior=0, the value returned for DATA_TYPE is -15 (NCHAR). If JDBCBehavior=1, the value returned is 1 (CHAR).

TYPE_NAME = long varchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = long varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (DB2 for Linux/UNIX/Windows), 32704 (DB2 for z/OS), 32700 (DB2 for i) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = long varchar for bit data AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = long varchar for bit data MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (Db2 for Linux/UNIX/Windows), 32698 (Db2 for z/OS), 32739 (Db2 for i) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = long vargraphic AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = -1 (LONGVARCHAR) [12] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = longvarchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16352 (Db2 for z/OS), 16370 (Db2 for i), 16336 (Db2 for Linux/UNIX/Windows) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = numeric AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = (*precision*,*scale*) DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = numeric MAXIMUM_SCALE = 31	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX =10 PRECISION = 31 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = real AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float(4) MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 24 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

[12] If JDBCBehavior=0, the value returned for DATA_TYPE is -16 (LONGNVARCHAR). If JDBCBehavior=1 the value returned is -1 (LONGVARCHAR).

TYPE_NAME = rowid [13] AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = not null generated always DATA_TYPE = -2 (Binary) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = rowid MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = NULL PRECISION = 40 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = smallint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = time AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = false LITERAL_PREFIX = {t ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = time MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[13] Supported only for Db2 for z/OS and Db2 for i.

TYPE_NAME = timestamp AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = timestamp MAXIMUM_SCALE = 6	MINIMUM_SCALE = 6 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 26 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varbinary AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = VARBINARY(X' LITERAL_SUFFIX = ') LOCAL_TYPE_NAME = varbinary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32703 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = varchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = (*max length*) DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32672 (Db2 for Linux/UNIX/Windows), 32698 (Db2 for z/OS), 32739 (Db2 for i) SEARCHABLE = 3 (Db2 for Linux/UNIX/Windows), 1 (Db2 for z/OS), 1 (Db2 for i) SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar() for bit data AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = (*max length*) DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar() for bit data MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32672 (Db2 for Linux/UNIX/Windows), 32698 (Db2 for z/OS), 32739 (Db2 for i) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = vargraphic AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = 12 (VARCHAR) [14] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16352 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = xml [15] AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 2005 (CLOB) or 2009 (SQLXML) [16] FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = xml MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

SQL escape sequences

The driver supports the following SQL escape sequences.

- Date, Time, and Timestamp Escape Sequences
- Scalar Functions
- Outer Join Escape Sequences
- LIKE Escape Character Sequence for Wildcards

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.

[14] If JDBCBehavior=0, the value returned for DATA_TYPE is -9 (NVARCHAR). If JDBCBehavior=1, the value returned is 12 (VARCHAR).

[15] Supported only for Db2 for Linux/UNIX/Windows and Db2 for i V7R1 and higher.

[16] If JDBCBehavior=0, the value returned for DATA_TYPE is 2009 (SQLXML). If JDBCBehavior=1, the value returned is 2005 (CLOB). In addition, the XMLDescribeType property can override driver mappings.

Supported scalar functions

You can use scalar functions in SQL statements with the following syntax:

`{fn scalar-function}`

where:

scalar-function

is a scalar function supported by the driver, as listed in the following table.

Example:

```
SELECT id, name FROM emp WHERE name LIKE {fn UCASE('Smith')}
```

Refer to "Scalar functions" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

Table 3: Supported Scalar Functions for Db2 for Linux, UNIX, and Windows

String Functions	Numeric Functions	Timedate Functions	System Functions
ASCII	ABS or ABSVAL	CURRENT_DATE	COALESCE
BLOB	ACOS	CURRENT_TIME	DEREF
CHAR	ASIN	CURRENT_TIMESTAMP	DLCOMMENT
CHAR_LENGTH	ATAN	DATE	DLLINKTYPE
CHR	ATAN2	DAY	DLURLCOMPLETE
CLOB	BIGINT	DAYNAME	DLURLPATH
CONCAT	CEILING or CEIL	DAYOFMONTH	DLURLPATHONLY
DBCLOB	COS	DAYOFWEEK	DLURLSCHEME
DIFFERENCE	COT	DAYOFYEAR	DLURLSERVER
GRAPHIC	DECIMAL	DAYS	DLVALUE
HEX	DEGREES	CURDATE	EVENT_MON_STATE
INSERT	DIGITS	CURTIME	GENERATE_UNIQUE
LCASE or LOWER	DOUBLE	EXTRACT	IFNULL
LCASE[17]	EXP	HOUR	NODENUMBER
LEFT	FLOAT	JULIAN_DAY	NULLIF
LENGTH	FLOOR	MICROSECOND	PARTITION
LOCATE	INTEGER	MIDNIGHT_SECONDS	RAISE_ERROR
LONG_VARCHAR	LN	MINUTE	TABLE_NAME
LONG_VARGRAPHIC	LOG	MONTH	TABLE_SCHEMA
LTRIM		MONTHNAME	TRANSLATE
OCTET_LENGTH		NOW	TYPE_ID

[17] SYSFUN schema.

String Functions	Numeric Functions	Timedate Functions	System Functions
POSSTR	LOG10	QUARTER	TYPE_NAME
REPEAT	MOD	SECOND	TYPE_SCHEMA
REPLACE	POWER	TIME	VALUE
RIGHT	RADIANS	TIMESTAMP	
RTRIM	RAND	TIMESTAMPADD	
SOUNDEX	REAL	TIMESTAMP_ISO	
SPACE	ROUND	TIMESTAMPDIFF	
SUBSTR	SIGN	WEEK	
SUBSTRING	SIN	YEAR	
TRUNCATE or TRUNC	SMALLINT		
UCASE or UPPER	SQRT		
VARCHAR	TAN		
VARGRAPHIC	TRUNCATE		
	ATANH		
	COSH		
	SINH		
	TANH		

Table 4: Supported Scalar Functions for Db2 for z/OS

String Functions	Numeric Functions	Timedate Functions	System Functions
BLOB	ABS or ABSVAL	ALTDATE	COALESCE
CHAR	ACOS	ALTTIME	IFNULL
CLOB	ASIN	CURDATE	NULLIF
CONCAT	ATAN	CURRENT_DATE	RAISE_ERROR
DBCLOB	ATANH	CURRENT_TIME	TABLE_NAME
GRAPHIC	ATAN2	CURRENT_TIMESTAMP	TABLE_LOCATION
HEX	BIGINT	CURTIME	TABLE_SCHEMA
INSERT	CEILING or CEIL	DATE	TRANSLATE
LCASE or LOWER	COS	DAY	VALUE
LCASE[17] [18]	COSH	DAYOFMONTH	
LEFT	CURRENCY	DAYOFWEEK	
LENGTH		DAYOFYEAR	

[18] SYSFUN schema.

String Functions	Numeric Functions	Timedate Functions	System Functions
LOCATE	DEC	DAYS	
LTRIM	DECIMAL	HOUR	
POSSTR	DEGREES	JULIAN_DAY	
REPEAT	DIGITS	MICROSECOND	
REPLACE	DOUBLE	MIDNIGHT_SECONDS	
RIGHT	DOUBLE_PRECISION	MINUTE	
RTRIM	EXP	MONTH	
SPACE	FLOAT	MONTHNAME	
SUBSTR	FLOOR	NOW	
STRIP	INT	QUARTER	
TRUNCATE or TRUNC	INTEGER	SECOND	
UCASE or UPPER	LN	TIME	
VARCHAR	LOG	TIMESTAMP	
VARGRAPHIC	LOG10	TIMESTAMPADD	
	MOD	WEATHER	
	POWER	WEEK	
	RADIANS	YEAR	
	RAND		
	REAL		
	ROUND		
	ROWID		
	SIGN		
	SIN		
	SINH		
	SMALLINT		
	SQRT		
	TAN		
	TANH		
	TRUNCATE		

Table 5: Supported Scalar Functions for Db2 for i

String Functions	Numeric Functions	Timedate Functions	System Functions
BLOB CHAR CLOB CONCAT DBCLOB GRAPHIC HEX INSERT LCASE or LOWER LCASE[17] [19] LEFT LENGTH LOCATE LTRIM POSSTR RIGHT RTRIM SPACE SUBSTR STRIP TRUNCATE or TRUNC UCASE or UPPER VARCHAR VARGRAPHIC	ABS or ABSVAL ACOS ASIN ATAN ATANH ATAN2 BIGINT CEILING or CEIL COS COT COSH CURRENCY DEC DECIMAL DEGREES DIGITS DOUBLE DOUBLE_PRECISION EXP FLOAT FLOOR INT INTEGER LN LOG LOG10 MOD POWER RADIANS RAND REAL	ALTDATE ALTTIME CURDATE CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP CURTIME DATE DAY DAYOFMONTH DAYOFWEEK DAYOFYEAR DAYS HOUR JULIAN_DAY MICROSECOND MIDNIGHT_SECONDS MINUTE MONTH NOW QUARTER SECOND TIME TIMESTAMP TIMESTAMPADD WEATHER WEEK YEAR	COALESCE IFNULL NULLIF RAISE_ERROR TABLE_NAME TABLE_LOCATION TABLE_SCHEMA TRANSLATE VALUE

[19] SYSFUN schema.

String Functions	Numeric Functions	Timedate Functions	System Functions
	ROUND ROWID SIGN SIN SINH SMALLINT SQRT TAN TANH TRUNCATE		

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:

- Interactive SQL
- Tableau
- 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 db2.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
   ```

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 INFORMATION_SCHEMA.SYSTEM_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**.

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:

   ```
   db2.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 example shows how to connect using the user ID/password authentication.

     ```
     jdbc:datadirect:db2://myserver:50000;Database=account;
     ```

     Note: See Authentication on page 53 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **Username**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.
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\db2.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/db2.jar
   ```

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

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

  ```
  Db2
  ```

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

  ```
  jdbc:datadirect:db2:
  ```

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

  ```
  com.ddtek.jdbc.db2.DB2Driver
  ```

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 example shows how to connect using the user ID/password authentication.

     ```
     jdbc:datadirect:db2://myserver:50000;
     Database=account;User=test;Password=secret;
     ```

     Note: See Authentication on page 53 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 BLOGS table:

```
SELECT * FROM BLOGS
```

You have successfully accessed your data with DbVisualizer.

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:

- Performance considerations

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/db2.jar`

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\db2.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/db2.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 example shows, this method specifies a string containing a connection URL.

User ID/password authentication

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:db2://myserver:50000;
    Database=account;User=test;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.

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/password authentication.

Connection URL Syntax

The connection URL takes the following form:

For Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud

```
jdbc:datadirect:db2://servername:port;
Database=databasename;User=userID;
Password=password;[property=value[;...]];
```

For Db2 for i and Db2 for z/OS

```
jdbc:datadirect:db2://servername:port;
LocationName=locationname;User=userID;
Password=password;[property=value[;...]];
```

where:

servername

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

port

is the number of the TCP/IP port.

databasename

is the name of the database to which you want to connect.

locationname

is the name of the DB2 location that you want to access.

userID

specifies the user ID that is used to connect to the Db2 database.

password

specifies a password that is used to connect to your Db2 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:db2://myserver1:50000;
    Database=account;User=jsmith;Password=secret;");


Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:db2://myserver2:446;
    LocationName=sample;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

Connection property descriptions on page 95
Connection URL examples on page 14
Authentication on page 53

Testing the connection

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

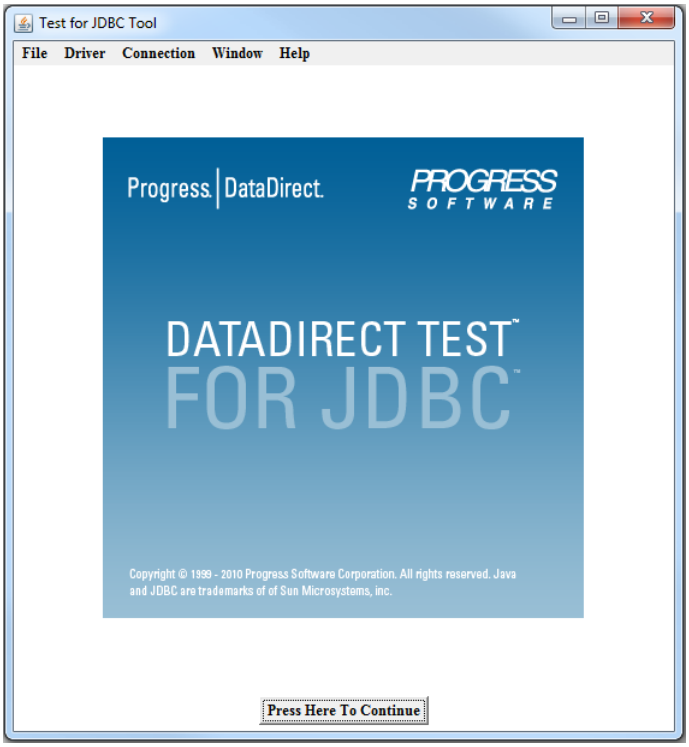
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC_60\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC_60/testforjdbc`

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

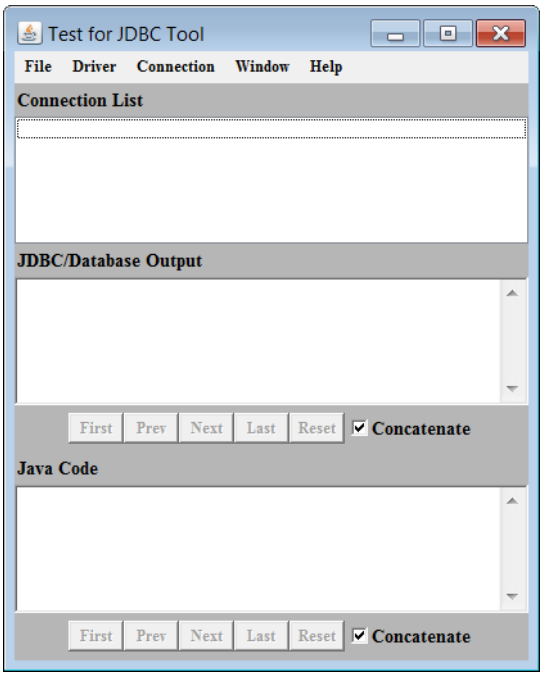
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



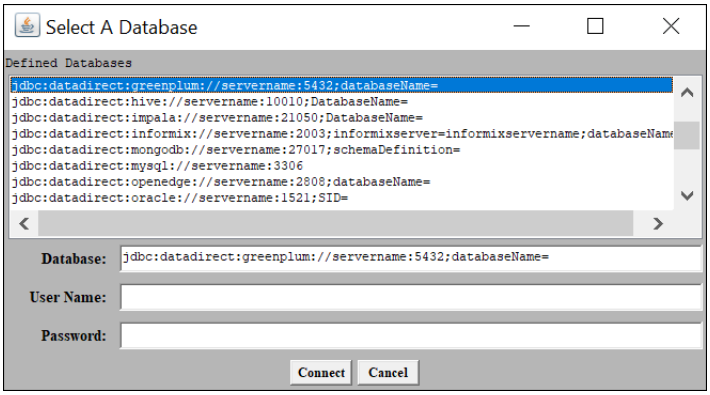
3. Click **Press Here to Continue**.

   The main dialog appears:



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

   The **Select A Database** dialog appears:



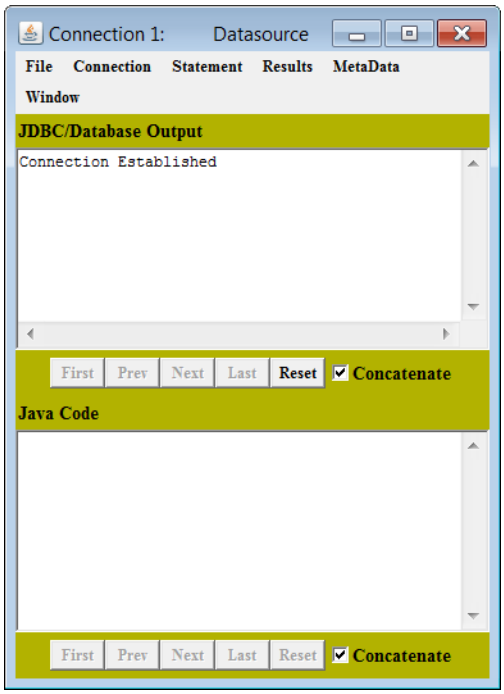
5. Select the appropriate database template from the **Defined Databases** field.
6. In the **Database** field, specify all required connection properties.

   For example:

   ```
   jdbc:datadirect:db2://myserver:50000;Database=accounts;
   ```

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 data source 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.

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.db2.DB2DataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required to establish connection:

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").

```
DB2DataSource mds = new DB2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabase("payroll");
```

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 instance of DB2DataSource)
{
DB2DataSource jmds = (DB2DataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
System.out.println("database=" + jmds.getDatabase());
```

```
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.

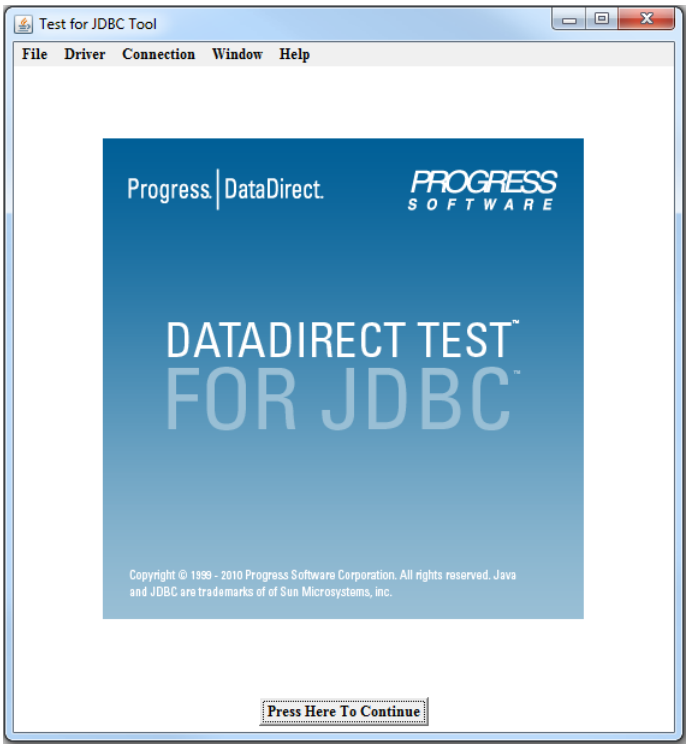
Take the following steps to establish a connection.

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.

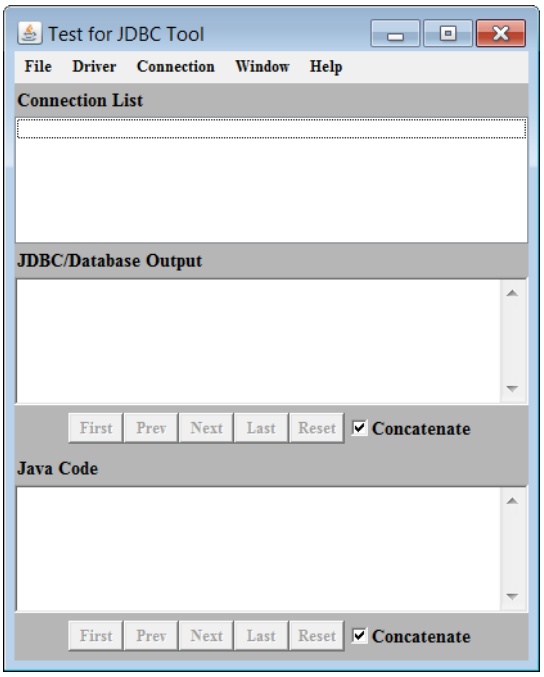
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



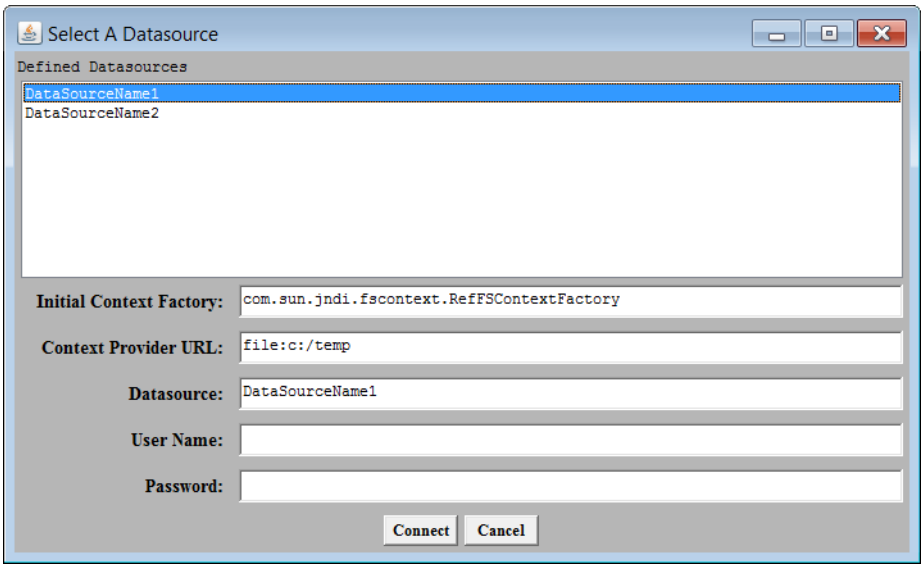
3. Click **Press Here to Continue**.

   The main dialog appears:



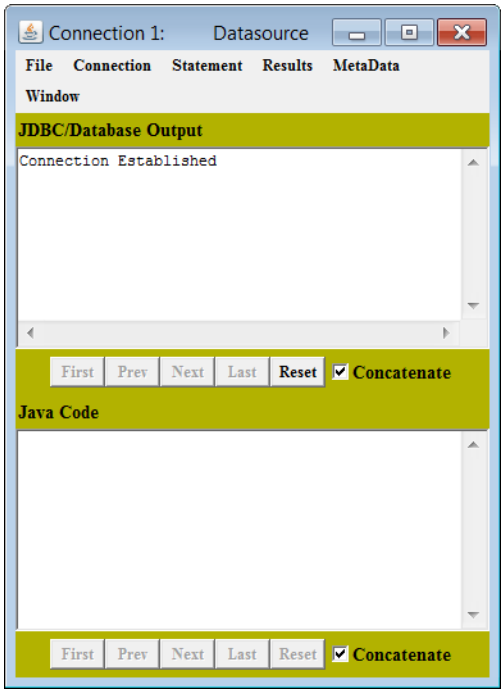
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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 Db2 driver supports the following methods of authentication.

- *User ID/password authentication* authenticates the user to the database using a user ID and password. Depending on the method you specify, the driver passes one of the following sets of credentials to the Db2 database server for authentication.
  - Encrypted user ID and password
  - User ID in clear text and an encrypted password
  - Both user ID and password in clear text
- *Kerberos authentication* uses Kerberos, a trusted third-party authentication service, to verify user identities. Kerberos authentication can take advantage of the user ID and password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application.

  This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos and MIT Kerberos.
- *Client authentication* uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The Db2 database server relies on the client to authenticate the user and does not provide additional authentication.

  Because the database server does not authenticate the user, use this method of authentication if you can guarantee that only trusted clients can access the database server.
- *GSS plug-in authentication* uses a set of APIs that you can customize to suit your security requirements. Db2 provides the required infrastructure for implementing custom security plug-ins along with some default security plug-ins.

The driver's AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections.

User ID/password authentication

To configure the driver to use user ID/password authentication:

- Set the AuthenticationMethod connection property to one of the following valid user ID/ password values. The default is `clearText`.
  - `clearText`: The driver sends the user ID and password in clear text to the Db2 server for authentication.
  - `encryptedPassword`: The driver sends a clear text user ID and an encrypted password to the Db2 server for authentication.
  - `encryptedPasswordAES`: The driver sends a clear text user ID and an AES-encrypted password to the Db2 server for authentication.
  - `encryptedUIDPassword`: The driver sends an encrypted user ID and password to the Db2 server for authentication.
  - `encryptedUIDPasswordAES`: The driver sends an AES-encrypted user ID and password to the Db2 server for authentication.

  Note: If you enable AES encryption by specifying `encryptedPasswordAES` or `encryptedUIDPasswordAES`, you can configure random number generator secure seeding. If you want to configure secure seeding, complete your configuration of user id/password authentication by proceeding to "Random number generator secure seeding".

  Note: For more information on the user ID/password values, see "AuthenticationMethod".

- Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud.
- Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for i.
- Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
- Set the Password property to specify the password.
- Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.
- Set the User property to specify the user name that is used to connect to the database.

The following examples show the connection information required to connect to a Db2 for Linux, UNIX, and Windows database using user ID/password authentication.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:db2://myserver:50000;DatabaseName=payroll;
 User=test;Password=secret);
```

Data Source

```
Db2DataSource mds = new Db2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
```

```
mds.setPortNumber("50000");
mds.setDatabaseName("payroll");
mds.setUser("jsmith");
mds.setPassword("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

Random number generator secure seeding on page 55
AuthenticationMethod on page 111

Random number generator secure seeding

Db2 uses a random number generator for secure seeding of data encrypted with the Advanced Encryption Standard (AES) algorithm. If you have enabled AES encryption with the AuthenticationMethod connection property, you should consider how best to implement secure seeding in your environment. The driver supports random number generator implementations by way of the RandomGenerator and SecureRandomAlgorithm connection properties. The RandomGenerator connection property allows you to specify the type of random number generator the database uses for secure seeding. If you select a cryptographically strong number generation algorithm, you can then use the SecureRandomAlgorithm connection property to specify any number generation algorithm included in the JDK packaged with your system.

Note: When establishing a connection with a connection string, RandomGenerator and SecureRandomAlgorithm should precede the User and Password connection properties in the connection URL. When using a data source connection, RandomGenerator and SecureRandomAlgorithm should be set before making calls to `setUser()`, `setPassword()`, or `setNewPassword()`.

The following steps outline how to configure a random number generator for secure seeding.

1. Configure the basic connection properties required for a connection:
   - Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud.
   - Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for I.
   - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
   - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.
2. If suitable to your environment, enable AES encryption by setting the AuthenticationMethod property to `encryptedPasswordAES` or `encryptedUIDPasswordAES`.
3. Set the RandomGenerator connection property.
   - If you specify `random`, no further steps are required. A stream of pseudorandom numbers will be generated for secure seeding, and you have completed driver configuration of the random number generator.
   - If you specify `secureRandom`, proceed to the next step.

4. Set the SecureRandomAlgorithm connection property by specifying the name of the SecureRandom number generation algorithm as a string. For example, `SecureRandomAlgorithm=SHA1PRNG`.

   Note: Refer to your database management system documentation to see which SecureRandom number generation algorithms are included in the JDK packaged with your system. Additional information is also available on the Java Cryptography Architecture Standard Algorithm Name Documentation for JDK 8 Web page.

5. Set the User property to specify the user name that is used to connect to the database.
6. Set the Password property to specify the password.

The following examples show the connection information required to connect to a Db2 for Linux, UNIX, and Windows database using user ID/password authentication and AES encryption.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:db2://myserver:50000;DatabaseName=payroll;
 AuthenticationMethod=encryptedUIDPasswordAES;RandomGenerator=random;
 SecureRandomAlgorithm=SHA1PRNG;User=test;Password=secret);
```

Data Source

```
Db2DataSource mds = new Db2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabaseName("payroll");
mds.setAuthenticationMethod("encryptedUIDPasswordAES");
mds.setRandomGenerator("random");
mds.setSecureRandomAlgorithm("SHA1PRNG");
mds.setUser("jsmith");
mds.setPassword("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

AuthenticationMethod on page 111
RandomGenerator on page 155
SecureRandomAlgorithm on page 158
User ID/password authentication on page 54
Performance considerations on page 73

Kerberos authentication

This section provides requirements and instructions for configuring Kerberos authentication for the Db2 driver.

See also

Permissions for Kerberos authentication on page 60
AuthenticationMethod on page 111

Product requirements for Kerberos

Your environment must meet the requirements listed in the following table to implement Kerberos authentication with the driver.

Note: The domain controller must administer both the database server and the client.

Table 6: Kerberos Authentication Requirements for the Db2 Driver

Component	Requirements
Database server	The database server must be running one of the following database versions: • Db2 for Linux/UNIX/Windows • Db2 for z/OS
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC. Network authentication must be provided by one of the following methods: • Windows Active Directory on one of the following operating systems: • Windows Server • MIT Kerberos 1.5 or higher
Client	J2SE 8 or higher must be installed.

See also

Configuring the driver on page 57

Configuring the driver

The driver supports Kerberos authentication. Note that the driver no longer sets the java.security.krb5.conf system property to force the use of the krb5.conf file installed with the driver jar files in the /lib directory of the product installation directory.

Important: A properly configured Kerberos environment must include a means of obtaining a Kerberos Ticket Granting Ticket (TGT). For a Windows Active Directory implementation, Active Directory automatically obtains the TGT. However, for a non-Active Directory implementation, the means of obtaining the TGT must be automated or handled manually.

To configure the driver to user Kerberos:

1. Set the driver's AuthenticationMethod property to `auto` (default) or `kerberos`. See "AuthenticationMethod" for more information about setting a value for this property.
2. Specify the JAAS login module in your JAAS login configuration file using either of the following methods.
   - Modify the `JDBC_DRIVER_01` entry in the `JDBCDriverLogin.conf` file to include the JAAS login module information needed for your environment. The JDBCDriverLogin.conf file is installed in the /lib directory of the driver installation directory.

- Specify a JAAS login configuration file directly in your application with the java.security.auth.login.config system property. The specified login configuration file must contain the JAAS login module information with the entry JDBC_DRIVER_01.

Whether you are using the `JDBCDriverLogin.conf` file or another file, the login configuration file must contain the entry JDBC_DRIVER_01 with JAAS login module information. The following examples show that the JAAS login module information depends on your JRE.

Oracle JRE

```
JDBC_DRIVER_01 {com.sun.security.auth.module.Krb5LoginModule
required useTicketCache=true;};
```

IBM JRE Example

```
JDBC_DRIVER_01 {com.ibm.security.auth.module.Krb5LoginModule
required useDefaultCcache=true;};
```

3. Set the default realm name and the KDC name for that realm using either of the following methods. (If using Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.)

   - Modify the `krb5.conf` file to include the default realm name and the KDC name for that realm. For example, if your Kerberos realm name is `XYZ.COM` and your KDC name is kdc1, your `krb5.conf` file would include the following entries.

     ```
     [libdefaults]
        default_realm = XYZ.COM
     [realms]
        XYZ.COM = {
        kdc = kdc1
        }
     ```

     Note: During installation, a `krb5.conf` file is installed in the `/lib` directory of the product installation directory. The installed `krb5.conf` name for that realm. If you are not already using another `krb5.conf` file for your Kerberos implementation, you can modify it to suit your environment. However, you will either need to specify the location of this file using the java.security.krb5.conf system property, or you will need to add the file to a directory where it may be found by your JVM. See "Keberos Requirements" in your Java documentation for details on the algorithm used to locate the `krb5.conf` file.

   - Specify the Java system properties, java.security.krb5.realm and java.security.krb5.kdc, in your application. For example, if the default realm name is `XYZ.COM` and the KDC name is `kdc1`, your application would include the following settings.

     ```
     System.setProperty("java.security.krb5.realm","XYZ.COM");

     System.setProperty("java.security.krb5.kdc","kdc1")
     ```

     Note: Even if you do not use the `krb5.conf` file to specify the realm and KDC names, you may need to modify your `krb5.conf` file to suit your environment. Refer to your database vendor documentation for detailed information.

4. For Java SE 13 and higher, set the GSS client library to be used when communicating with the KDC. By default, the driver uses the GSS library and mechanisms provided by the JDK. However, you can also use the native GSS library for your platform by configuring the following Java system properties as described:

Important: If you are using Windows Defender Credential Guard, you must set the Java system properties as described in this step.

- Set `sun.security.jgss.native` to `true`.
- For Microsoft SSPI, set `javax.security.auth.useSubjectCredsOnly` to `false`.
- Optionally, set `sun.security.jgss.lib` to specify the absolute path of the native library file. If you do not provide a value, the JVM will load the default GSS library file for the platform.

  Note: Starting with Java SE 13, the native Windows interface will be Microsoft SSPI, and the GSS client library will be the `sspi.bridge.dll` file.

5. If using Kerberos authentication with a Security Manager on a Java Platform, you must grant security permissions to the application and driver. See "Permissions for Kerberos authentication" for an example.

See also

AuthenticationMethod on page 111
Product requirements for Kerberos on page 57
Permissions for Kerberos authentication on page 60

Specifying user credentials for Kerberos authentication (delegation of credentials)

By default, when Kerberos authentication is used, the Db2 driver takes advantage of the user name and password maintained by the operating system to authenticate users to the database. By allowing the database to share the user name and password used for the operating system, users with a valid operating system account can log into the database without supplying a user name and password.

Many application servers or Web servers act on behalf of the client user logged on the machine on which the application is running, rather than the server user. If you want the driver to use user credentials other than the server user's operating system credentials, include code in your application to obtain and pass a `javax.security.auth.Subject` used for authentication as shown in the following example.

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;

//  The following code creates a javax.security.auth.Subject instance
//  used for authentication. Refer to the Java Authentication
//  and Authorization Service documentation for details on using a
//  LoginContext to obtain a Subject.

LoginContext lc = null;
Subject subject = null;

try {
   lc = new LoginContext("JaasSample", new TextCallbackHandler());
   lc.login();
   subject = lc.getSubject();
}
catch (Exception le) {
   ... // display login error
}

// This application passes the javax.security.auth.Subject
```

```
// to the driver by executing the driver code as the subject

Connection con =
   (Connection) Subject.doAs(subject, new PrivilegedExceptionAction() {

   public Object run() {

      Connection con = null;
      try {

         Class.forName("com.ddtek.jdbc.db2.DB2Driver");
         String url = "jdbc:datadirect:db2://myServer:50000;DatabaseName=jdbc";
         con = DriverManager.getConnection(url);
         }
      catch (Exception except) {
      ... //log the connection error
      Return null;
      }

       return con;
   }
});

// This application now has a connection that was authenticated with
// the subject. The application can now use the connection.

Statement stmt = con.createStatement();
String sql = "SELECT * FROM employee";
ResultSet rs = stmt.executeQuery(sql);

... // do something with the results
```

See also

Kerberos authentication on page 56

Permissions for Kerberos authentication

If using Kerberos authentication with a Security Manager on a Java Platform, you must grant security permissions to the application and driver.

To grant security permission for the Db2 driver, add the following code to the Java security policy file.

```
grant codeBase "file:/install_dir/lib/60-" {
   permission javax.security.auth.AuthPermission
      "createLoginContext.DDTEK-JDBC";
   permission javax.security.auth.AuthPermission "doAs";
   permission javax.security.auth.kerberos.ServicePermission
      "krbtgt/your_realm@your_realm", "initiate";
   permission javax.security.auth.kerberos.ServicePermission
      "principal_name/db_hostname@your_realm", "initiate";
};
```

where:

install_dir

is the product installation directory.

your_realm

is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

principal_name

is the service principal name registered with the Key Distribution Center (KDC) that identifies the database service.

db_hostname

is the host name of the machine running the database.

See also

Configuring the driver on page 57

Client authentication

When client authentication is enabled (`AuthenticationMethod=client`), the driver uses the user ID of the user logged onto the system on which the driver is running when establishing a connection. The Db2 database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any values specified by the User and Password properties.

To configure the driver to use client authentication:

- Set the AuthenticationMethod connection property to `client`.
- Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud.
- Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for i.
- Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
- Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.

The following examples show the connection information required to connect to a Db2 for Linux, UNIX, and Windows database using client authentication.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:db2://myserver:50000;DatabaseName=payroll
  AuthenticationMethod=client);
```

Data Source

```
Db2DataSource mds = new Db2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabaseName("payroll");
mds.setAuthenticationMethod("client");
```

See also

AuthenticationMethod on page 111

GSS plug-in authentication

Db2 supports the Generic Security Standard (GSS) plug-in for authentication. The GSS plug-in provides a generic interface that eliminates the need to write your application for specific security implementations based on platform, security mechanism, or transfer protocol. The plug-in consists of a set of APIs that you can implement to customize your authentication requirements and interoperate with various security methods.

To configure a GSS plug-in for authentication.

- Configure the basic connection properties required for a connection:
  - Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud
  - Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for I.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.
- Set the AuthenticationMethod connection property to `pluginSecurity`.
- Set the GSSPluginName connection property to the plug-in name enabled on the server. The plug-in name is case-sensitive.
- Set the GSSPluginObject connection property to a valid object of a class that extends the `com.ddtek.jdbc.db2.gssplugin.DB2GSSPluginClient` class and provides a valid implementation for the `getTicket()` method. This value must be a data source object or a java.util.Properties object supplied through the DriverManager class.

Note: The implementation of the GSSPluginObject property depends on the type of GSS APIs enabled on the Db2 server. Both the server and the client must have the same plug-in enabled. The server must have `pluginSecurity` configured.

The following examples show the connection information required to establish a session using the GSS plug-in authentication.

Connection URL

```
jdbc:datadirect:db2://serverName:60000;databaseName=dbName;
java.util.Properties properties = new java.util.Properties();
properties.put("authenticationMethod", "pluginSecurity");
properties.put("user", "userName");
properties.put("password", "password");
properties.put("GSSPluginName", "gssapi_name");
properties.put("GSSPluginObject", new DB2GSSPlugin());
Connection connection= DriverManager.getConnection(connectionUrl, properties);
```

Data Source

```
DB2DataSource dataSource = new DB2DataSource();
dataSource.setServerName("serverName");
dataSource.setPortNumber(60000);
dataSource.setDatabaseName("dbName");
dataSource.setUser("userName");
dataSource.setPassword("password");
dataSource.setAuthenticationMethod("pluginSecurity");
```

```
dataSource.setGSSPluginName("gssapi_name");
dataSource.setGSSPluginObject(new DB2GSSPlugin());
Connection connection = dataSource.getConnection();
```

See also

AuthenticationMethod on page 111
GSSPluginName on page 132
GSSPluginObject on page 133

Implementing the DB2GSSPluginClient class

To implement the GSS plug-in, you need to write a GSS-API plug-in class that extends the abstract `com.ddtek.jdbc.db2.gssplugin.DB2GSSPluginClient` class provided by the driver. Then you need to provide an instance of that class as a value for the GSSPluginObject property. The source code for the abstract class `com.ddtek.jdbc.db2.gssplugin.DB2GSSPluginClient` is shown in the following example.

```
package com.ddtek.jdbc.db2.gssplugin;

import java.sql.SQLException;
import org.ietf.jgss.GSSContext;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSException;

//  The following code creates an abstract class for plugin client authentication
//  implementation. The client application must extend this class and provide
//  implementation for the getTicket() method. With pluginSecurity authenticationMethod,
//  the driver calls the getTicket method during login packet communication
//  and sends the generated GSS ticket to the DBMS.

public abstract class DB2GSSPluginClient
{
 protected GSSContext gssContext;
 protected String serverPrincipalName;
 protected GSSCredential gssCredential;

// The following method establishes the security context between the driver and the DBMS.
// The driver calls this method to obtain a GSS token from the client security
// implementation for the username/password pair. The GSS token received is sent to
// the DBMS. The first call to the method should specify null. Subsequent calls must
// specify the token obtained from the DBMS.
// @param userid - user Id for database authentication. The value can be null.
// @param password - password for database authentication. The value can be null.
// @param serverToken - The token obtained from the DBMS. The value is null.
// @throws SQLException - Exception occurred during GSS token generation wrapped
// within SQLException object.
// @return byte[] - The GSS token obtained from the client security implementation
// that is sent to the DBMS.

public abstract byte[] getTicket(String userid,String password,byte[] serverToken)
throws SQLException;

// The following code releases any system resources and cryptographic information
// stored in the context object and invalidates the context.

public void cleanup() throws GSSException
 {
  if (gssCredential != null)
      gssCredential.dispose();

  if (gssContext != null)
   gssContext.dispose();
 }
```

```
// Getter for gssContext

public GSSContext getGssContext()
 {
  return gssContext;
 }

// Setter for gssContext

public void setGssContext(GSSContext gssContext)
 {
  this.gssContext = gssContext;
 }

// Getter for serverPrincipalName

public String getServerPrincipalName()
 {
  return serverPrincipalName;
 }

// Setter for serverPrincipalName

public void setServerPrincipalName(String serverPrincipalName)
 {
  this.serverPrincipalName = serverPrincipalName;
 }

// Getter for gssCredential

public GSSCredential getGssCredential()
 {
  return gssCredential;
 }

// Setter for gssCredential

public void setGssCredential(GSSCredential gssCredential)
 {
  this.gssCredential = gssCredential;
 }
}
```

See also

GSS plug-in authentication on page 62

Data encryption

The Db2 driver supports database-specific encryption (Db2's own encryption protocol) for the following databases:

- Db2 for Linux/UNIX/Windows
- Db2 for z/OS

The Db2 driver supports TLS/SSL encryption for the following databases:

- Db2 for Linux/UNIX/Windows
- Db2 for z/OS
- Db2 for i
- Db2 Hosted

- Db2 Warehouse on Cloud

Configuring Db2-specific encryption

To configure the driver for Db2-specific encryption:

- Set the AuthenticationMethod property to `clearText`, `encryptedPassword`, `encryptedPasswordAES`, `encryptedUIDPassword`, or `encryptedUIDPasswordAES`.
- Set the EncryptionMethod property to `DBEncryption` or `RequestDBEncryption`.

Configuring TLS/SSL encryption

The following steps outline how to configure TLS/SSL encryption.

Note: Connection hangs can occur when the driver is configured for TLS/SSL and the database server does not support TLS/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 TLS/SSL.

To configure TLS/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.

- Configure the basic connection properties required for a connection:
  - Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud
  - Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for I.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.
- Set the EncryptionMethod property to `SSL`.
- For TLS/SSL server authentication, set the following properties or their corresponding Java system properties to specify the location and password of the truststore file.
  - TrustStore (javax.net.ssl.trustStore)
  - TrustStorePassword (javax.net.ssl.trustStorePassword)
- Optionally, set the CryptoProtocolVersion property to specify acceptable cryptographic protocol versions supported by your server. The default value is `TLSv1.3`.
- Optionally, set the ValidateServerCertificate property to `true` or `false`. When it is set to `true`, the driver validates the certificates sent by the database server. When it is set to `false`, the driver does not validate the certificates sent by the database server. The default value is `true`.

- Optionally, 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.
- If your database server is configured for TLS/SSL client authentication, configure your keystore information:
  - Set the KeyStore and KeyStorePassword properties or their corresponding Java system properties (javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword, respectively) to specify the location and password of the keystore file.
  - If any key entry in the keystore file is password-protected, set the KeyPassword property to the key password.

The following examples configure the driver to use TLS/SSL server and client encryptions with user ID/password authentication.

Connection URL

TLS/SSL server encryption:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:db2://myserver:50000;DatabaseName=payroll;
  EncryptionMethod=ssl;TrustStore=TrustStorePath;
  TrustStorePassword=secrettruststore;User=jsmith;
  Password=secret;);
```

TLS/SSL client encryption:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:db2://myserver:50000;DatabaseName=payroll;
  EncryptionMethod=ssl;KeyStore=KeyStorePath;
  KeyStorePassword=secretkeystore;User=jsmith;
  Password=secret;);
```

Data Source

TLS/SSL server encryption:

```
Db2DataSource mds = new Db2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabaseName("payroll");
mds.setEncryptionMethod("ssl");
mds.setTrustStore("TrustStorePath");
mds.setTrustStorePassword("secrettruststore");
mds.setUser("jsmith");
mds.setPassword("secret");
```

TLS/SSL client encryption:

```
Db2DataSource mds = new Db2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabaseName("payroll");
mds.setEncryptionMethod("ssl");
mds.setKeyStore("KeyStorePath");
mds.setKeyStorePassword("secretkeystore");
mds.setUser("jsmith");
mds.setPassword("secret");
```

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.

Limitations

Because certain algorithms used by Db2 are not FIPS compliant, the following User ID and password authentication methods are not supported in a FIPS enabled environment:

- `encryptedPassword`: The driver sends a clear text user ID and an encrypted password to the Db2 server for authentication.
- `encryptedPasswordAES`: The driver sends a clear text user ID and an AES-encrypted password to the Db2 server for authentication.
- `encryptedUIDPassword`: The driver sends an encrypted user ID and password to the Db2 server for authentication.
- `encryptedUIDPasswordAES`: The driver sends an AES-encrypted user ID and password to the Db2 server for authentication.

See "User ID/password authentication" for more information.

See also

User ID/password authentication on page 54

Proxy server

In some environments, your application may need to connect through a proxy server, for example, if your application accesses an external resource such as a Web service. At a minimum, your application needs to provide the following connection information when you invoke the JVM if the application connects through a proxy server:

- Server name or IP address of the proxy server
- Port number on which the proxy server is listening for HTTP/HTTPS requests

In addition, if authentication is required, your application may need to provide a valid user ID and password for the proxy server. Consult with your system administrator for the required information.

For example, the following command invokes the JVM while specifying a proxy server named `pserver`, a port of `808`, and provides a user ID and password for authentication:

```
java -Dhttp.proxyHost=pserver -Dhttp.proxyPort=808 -Dhttp.proxyUser=smith
-Dhttp.proxyPassword=secret -cp db2.jar com.acme.myapp.Main
```

Alternatively, you can use the ProxyHost, ProxyPort, ProxyUser, and ProxyPassword connection properties, but these properties are applied only for the first connection. See "Connection property descriptions" for details about these properties.

See also

Connection property descriptions on page 95
Connection URL examples on page 14

IP addresses

The driver supports Internet Protocol (IP) addresses in IPv4 and IPv6 formats.

The server name specified in a connection URL, or data source, can resolve to an IPv4 or IPv6 address. For example, the server name DB2Server in the following URL can resolve to either type of address:

```
jdbc:datadirect:db2://DB2Server:50000;DatabaseName=jdbc;User=test;
Password=secret
```

Alternatively, you can specify addresses using IPv4 or IPv6 format in the server name portion of the connection URL. For example, the following connection URL specifies the server using IPv4 format:

```
jdbc:datadirect:db2://123.456.78.90:50000;DatabaseName=jdbc;User=test;
Password=secret
```

You also can specify addresses in either format using the ServerName data source property. The following example shows a data source definition that specifies the server name using IPv6 format:

```
DB2DataSource mds = new DB2DataSource();
mds.setDescription("My DB2DataSource");
mds.setServerName("[ABCD:EF01:2345:6789:ABCD:EF01:2345:6789]");
mds.setPortNumber(50000);
...
```

Note: When specifying IPv6 addresses in a connection URL or data source property, the address must be enclosed by brackets.

In addition to the normal IPv6 format, the drivers support IPv6 alternative formats for compressed and IPv4/IPv6 combination addresses. For example, the following connection URL specifies the server using IPv6 format, but uses the compressed syntax for strings of zero bits:

```
jdbc:datadirect:db2://[2001:DB8:0:0:8:800:200C:417A]:50000;
DatabaseName=jdbc;User=test;Password=secret
```

Similarly, the following connection URL specifies the server using a combination of IPv4 and IPv6:

```
jdbc:datadirect:db2://[0000:0000:0000:0000:0000:FFFF:123.456.78.90]:
50000;DatabaseName=jdbc;User=test;Password=secret
```

For complete information about IPv6, go to the following URL:

http://tools.ietf.org/html/rfc4291#section-2.2

Failover

The driver provides the following levels of failover protection to ensure continuous, uninterrupted access to data.

- *Connection failover* provides failover protection for new connections only. The driver fails over new connections to an alternate, or backup, database server if the primary database server is unavailable, for example, because of a hardware failure or traffic overload. If a connection to the database is lost, or dropped, the driver does not fail over the connection. This failover method is the default.
- *Extended connection failover* provides failover protection for new connections and lost database connections. If a connection to the database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost, but not any work in progress.
- *Select Connection failover* provides failover protection for new connections and lost database connections. In addition, it provides protection for Select statements that have work in progress. If a connection to the

database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost and preserving the state of any work being performed by Select statements.

Note: For more information on connection failover and different levels of failover protection provided by the driver, refer to "Failover" in the *Progress DataDirect for JDBC Drivers Reference*.

To configure failover:

- Configure the basic connection properties required for a connection:
  - Set the DatabaseName property to specify the name of the database to which you want to connect. Valid only on Db2 for Linux, UNIX, and Windows; Db2 Hosted; and Db2 Warehouse on Cloud
  - Set the LocationName property to specify the name of the Db2 location that you want to access. Valid only on Db2 for z/OS and Db2 for I.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.
- Set the AlternateServers property to specify one or multiple alternate servers.

  Note: To turn off failover, do not specify a value for the AlternateServers property.

- Set the FailoverMode connection property to one of the following failover methods. The default is connection failover (`FailoverMode=connect`).
  - 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 performed by the last Select statement executed on the Statement object.
- If Extended Connection Failover (`FailoverMode=extended`) or Select Connection Failover (`FailoverMode=select`) is enabled, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection:
  - If set to `nonAtomic`, the driver continues with the failover process and posts any exceptions on the statement on which they occur. This is the default value.
  - 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.

- Optionally, set the ConnectionRetryCount property to specify the number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5 times.
- Optionally, set the ConnectionRetryDelay property to specify the number of seconds the driver waits between connection retry attempts. The default is 1second.
- Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (`FailoverPreconnect=false`).

The following examples configure the driver to use connection failover in conjunction with connection retry.

Connection URL

```
jdbc:datadirect:db2://myserver1:50000;Database=payroll;
User=jsmith;Password=secret;AlternateServers=(myserver2:50001;
Database=account,myserver3:50002;Database=sample);
ConnectionRetryCount=2;ConnectionRetryDelay=5;
```

In this example:

```
...myserver1:50000;Database=payroll...
```

is the part of the connection URL that specifies connection information for the primary server. Alternate servers are specified using the AlternateServers property. For example:

```
...;AlternateServers=(myserver2:50001;Database=account,myserver3:50002;Database=sample)
```

If a successful connection is not established on the Db2 driver's first pass through, the list of database servers (primary and alternate), the driver retries the list of servers in the same sequence twice (`ConnectionRetryCount=2`). Because the connection retry delay has been set to five seconds (`ConnectionRetryDelay=5`), the driver waits five seconds between retry passes.

Data Source

```
DB2DataSource mds = new DB2DataSource();
mds.setDescription("My Db2 Data Source");
mds.setServerName("myserver");
mds.setPortNumber("50000");
mds.setDatabase("payroll");
mds.setUser("jsmith");
mds.setPassword("secret");
mds.setAlternateServers("myserver2:50001;Database=account,myserver3:50002;Database=sample");
```

See also

Connection property descriptions on page 95

Client information

Many databases allow applications to store client information associated with a connection, which can be useful for database administration and monitoring purposes. The driver allows applications to store and return the following types of client information.

- Name of the application

- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Driver name and version

This information can feed directly into the Workload Manager (WLM) for workload management and monitoring purposes. See "Workload Manager (WLM)" for more information.

Refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Db2 Workload Manager (WLM) on page 86

Bulk Load

Because Db2 does not have native bulk load support, the driver emulates bulk load using the standard batch mechanism.

The BulkLoadBatchSize connection property affects how bulk load works with the Db2 driver. It suggests the number of rows to be loaded to the database when bulk loading data. The default is `2048`.

See also

BulkLoadBatchSize on page 113

Performance considerations

You can optimize application performance by adopting guidelines described in this section.

CatalogOptions: To improve performance, the driver can emulate getColumns() calls using the ResultSetMetaData object instead of querying database catalogs for the column information. Using emulation can improve performance because the SQL statement formulated by the emulation is less complex than the SQL statement formulated using getColumns(). The argument to getColumns() must evaluate to a single table. If it does not, because of a wildcard or null value, for example, the driver reverts to the default behavior for getColumns() calls.

CatalogSchema: To improve performance, views of system catalog tables can be created in a catalog schema other than the default. The Db2 driver can access the views of catalog tables if this property is set to the name of the schema containing the views. The default catalog schema is SYSCAT for Db2 for Linux/UNIX/Windows, SYSIBM for Db2 for z/OS, and QSYS2 for DB2 for i.

To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your Db2 database. See "Non-default schemas for catalog methods" for views for catalog tables that must exist in the specified schema.

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

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.

LobStreamingProtocol: In most cases, streaming provides better performance; however, when updating LOB data using Clob and Blob objects, the driver can materialize (cache) the data on the client, which can reduce network round trips to the database server and improve performance.

LongDataCacheSize: To improve performance when your application retrieves images, pictures, long text, binary data, or XML data, you can disable caching for long data on the client if your application retrieves long data column values in the order they are defined in the result set. If your application retrieves long data column values out or order, long data values must be cached.

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.

OptimizationProfile: An optimization profile contains optimization guidelines that are passed to Db2 to influence the generation of query plans for specific SQL statements. If you have tried other performance-tuning options, but you think that you are still getting less than optimal performance for a specific SQL statement, you can specify an optimization profile to provide explicit optimization guidelines. Refer to your Db2 documentation for details on creating an optimization profile.

RandomGenerator: By default, RandomGenerator is set to `secureRandom`. While `secureRandom` offers more secure seeding of random numbers, operations generally require additional processing with this configuration. Therefore, if your environment does not require more secure seeding, you should consider setting RandomGenerator to `random` to increase response times for your applications.

ResultSetMetaDataOptions: The driver’s performance may be adversely affected if you set this option to 1. If set to 1 and the ResultSetMetaData.getTableName method is called, the driver performs emulations which take additional processing.

SendStreamAsBlob: If the large binary objects you insert or update are stored as Blobs, performance can be improved by sending the binary stream as Blob data. In this case, this property should be set to `true`.

StripNewLines: If you know that the SQL statements used in your application do not contain newline characters, the driver can improve performance by omitting the parsing required to remove them.

UseCurrentSchema: If your application needs to access tables and views owned only by the current user, performance of your application can be improved by setting this property to `true`. When this property is set to `true`, the driver returns only tables and views owned by the current user when executing getTables() and getColumns() methods. Setting this property to `true` is equivalent to passing the user ID used on the connection as the schemaPattern argument to the getTables() or getColumns() call.

Refer to "Designing JDBC Applications for Performance Optimization" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using prepared statement pooling to optimize performance.

See also

Non-default schemas for catalog methods on page 82

4

Additional features and functionality

The following section describes additionally supported features and functionality that are specific to the driver.

For details, see the following topics:

- Db2 packages
- Returning and inserting/updating XML data
- Support for Db2 pureScale
- Non-default schemas for catalog methods
- Choosing a Db2 optimization class
- Reauthentication
- Db2 Workload Manager (WLM)
- Isolation levels
- Using scrollable cursors
- Db2 for Linux, UNIX, Windows stored procedure cursor type OUT parameters
- JTA support
- Large object (LOB) support
- Batch Inserts and Updates
- Parameter metadata support
- ResultSet metadata support

- Rowset support
- Auto-generated keys support
- TCP KeepAlive support

Db2 packages

The Db2 driver automatically creates all required Db2 packages the first time it connects to the database. By default, the Db2 packages contain 200 dynamic sections and are created in the NULLID collection (or library). You can override the default number of dynamic sections by setting the DynamicSections property. Similarly, you can override the collection in which the packages are created by setting the PackageCollection property.

Note: The initial connection may take a few minutes because of the number and size of the packages that must be created for the connection. Subsequent connections do not incur this delay. When the driver has completed creating packages, it writes the following message to the standard output: `DB2 packages created`.

In most cases, you do not need to create Db2 packages because the Db2 driver automatically creates them the first time it connects to the database. If you need to explicitly create them, you can create them using any of the following methods:

- Using the DataDirect Db2 Package Manager, a Java graphical tool that makes it easy for you to create, drop, and replace Db2 packages. See "Creating Db2 packages using the package manager" for instructions.
- Using Db2 driver connection properties specified in a connection URL or data source. See "Creating Db2 packages using connection properties" for instructions.
- Running the appropriate Db2 package creation list file for your operating system on the database server. See "Creating Db2 packages using package creation list files" for instructions.
- On z/OS and i only, copy the packages that are installed with the driver as described in "Copying the Db2 packages (z/OS and i)".

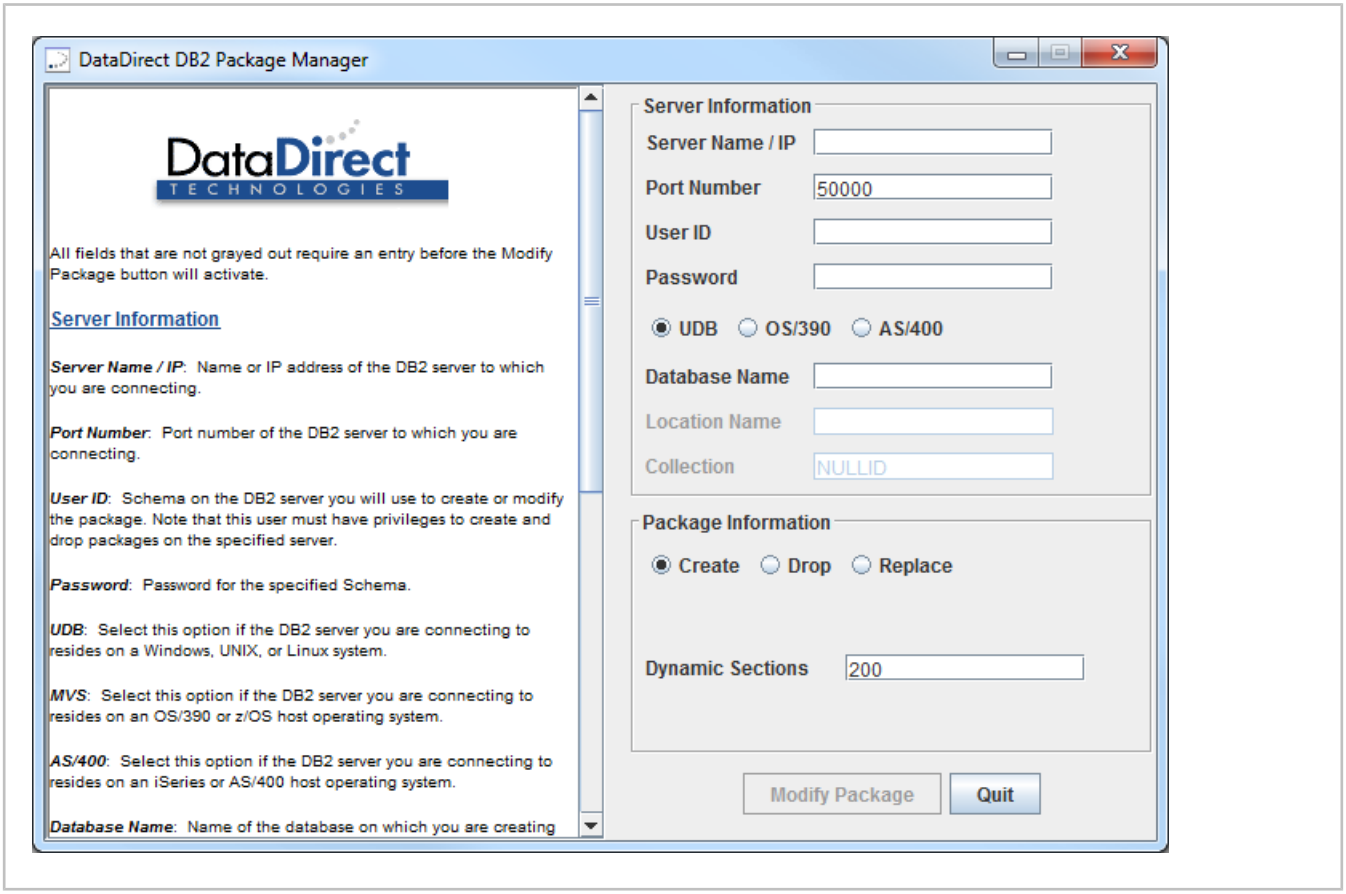
See also

Creating Db2 packages using the package manager on page 76
Creating Db2 packages using connection properties on page 77
Creating Db2 packages using package creation list files on page 78
Copying the Db2 packages (z/OS and i) on page 79

Creating Db2 packages using the package manager

Note: The user ID creating the Db2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To start the DataDirect DB2 Package Manager, run the `DB2PackageManager.bat` file (on Windows) or the `DB2PackageManager.sh` script (on UNIX) located in the `lib` subdirectory in the product installation directory. The DataDirect DB2 Package Manager dialog box appears.



Complete the fields in the right pane. Refer to the left pane of the DataDirect DB2 Package Manager for instructions on completing these fields. When you are satisfied with your choices, click the **Modify Package** button to create the package on the Db2 server.

Creating Db2 packages using connection properties

Note: The user ID creating the Db2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

The following table lists the connection properties you can use to explicitly create Db2 packages.

Table 7: Connection Properties for Creating Db2 Packages

Property	Database
`PackageCollection=` *`collection_name`* (where *`collection_name`* is the name of the collection or library to which Db2 packages are bound)	Db2 for z/OS and i
`CreateDefaultPackage=true`	Db2 for Linux/UNIX/Windows, z/OS, and i
`ReplacePackage=true`	Db2 for Linux/UNIX/Windows
`DynamicSections=` *x* (where *x* is a positive integer)	Db2 for Linux/UNIX/Windows, z/OS, and i

Note: To create new Db2 packages on Db2 for Linux/UNIX/Windows, you must use `ReplacePackage=true` in conjunction with `CreateDefaultPackage=true`. If a Db2 package already exists, it will be replaced when `ReplacePackage=true`.

Example: Db2 for Linux/UNIX/Windows

The following URL creates Db2 packages with 400 dynamic sections. If any Db2 packages already exist, they will be replaced by the new ones being created.

```
jdbc:datadirect:db2://server1:50000;DatabaseName=SAMPLE;
CreateDefaultPackage=TRUE;ReplacePackage=TRUE;DynamicSections=400
```

Example: Db2 for z/OS and i

The following URL creates Db2 packages with 400 dynamic sections.

```
jdbc:datadirect:db2://server1:446;LocationName=SAMPLE;
CreateDefaultPackage=TRUE;DynamicSections=400
```

Creating Db2 packages using package creation list files

Note: The user ID creating the Db2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To explicitly create Db2 packages, bind the appropriate package creation list files for your operating system on your database server (see the following table). When you bind the list files, if any Db2 packages exist, they will be replaced by the new packages. The list files create Db2 packages that contain 200 dynamic sections and are created in the NULLID collection.

Table 8: Db2 Package Creation List Files/Location

File	Location
DDJDBC_LUW.lst	*installdir* /DB2/bind/LUW
DDJDBC_400.lst	*installdir* /DB2/bind/iSeries
DDJDBC_MVS.lst	*installdir* /DB2/bind/zOS

To explicitly create Db2 packages:

1. Copy the appropriate list (`*.lst`) and bind (`*.bnd`) files to a directory on the database server.
2. From the directory on the database server where you placed the list and bind files, start the Db2 command-line utility.
3. Using the Db2 command-line utility, connect to the database where you want to bind the packages using the following command:

   `connect to` *database_name* `user` *authorization_name* `using` *password*

where:

database_name is the name of the database to which you are connecting.

authorization_name is the name of the user you are authenticating to the server.

password is the user's password.

4. Execute the Db2 bind command:

```
bind @ list_file grant public
```

where *list_file* is the name of the list file you want to bind.

Copying the Db2 packages (z/OS and i)

Note: The user ID creating the Db2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To copy the Db2 packages on z/OS and i operating systems, refer to the instruction file for your operating system in the following table:

Table 9: Instruction Files for Copying Db2 Packages

File	Location
CfJDBC zOS Manual Package Creation Instructions.txt	*installdir* \DB2\bind\zOS
CfJDBC AS400 Manual Package Creation Instructions.txt	*installdir* \DB2\bind\iSeries

The Db2 packages contain 200 dynamic sections.

Returning and inserting/updating XML data

For Db2 V9.1 and higher for Linux/UNIX/Windows, the Db2 driver supports the XML data type. Which JDBC data type the XML data type is mapped to depends on whether the JDBCBehavior and XMLDescribeType properties are set:

- If XMLDescribeType=clob or XMLDescribeType=blob, the driver maps the XML data type to the JDBC CLOB or BLOB data type, respectively, regardless of the setting of the JDBCBehavior property.
- If JDBCBehavior=1 (default) and the XMLDescribeType property is not set, the driver maps XML data to the JDBC CLOB data type.
- If JDBCBehavior=0 and the XMLDescribeType property is not set, XML data is mapped to SQLXML.

Returning XML data

You can specify whether XML data is returned as character or binary data by setting the XMLDescribeType property. For example, consider a database table defined as:

```
CREATE TABLE xmlTable (id int, xmlCol xml NOT NULL)
```

and the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
ResultSet rs=stmt.executeQuery(sql);
```

If your application uses the following connection URL, which specifies that the XML data type be mapped to the BLOB data type, the driver would return XML data as binary data:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;User=test;
Password=secret;XMLDescribeType=blob
```

Character data

When XMLDescribeType=`clob`, XML data is returned as character data. The result set column is described with a column type of CLOB and the column type name is xml.

When XMLDescribeType=`clob`, your application can use the following methods to return data stored in XML columns as character data:

- ResultSet.getString()
- ResultSet.getCharacterStream()
- ResultSet.getClob()
- CallableStatement.getString()
- CallableStatement.getClob()

The driver converts the XML data returned from the database server from the UTF-8 encoding used by the database server to the UTF-16 Java String encoding.

Your application can use the following method to return data stored in XML columns as ASCII data:

- ResultSet.getAsciiStream()

The driver converts the XML data returned from the database server from the UTF-8 encoding to the ISO-8859-1 (latin1) encoding.

Note: The conversion caused by using the getAsciiStream() method may create XML that is not well-formed because the content encoding is not the default encoding and does not contain an XML declaration specifying the content encoding. Do not use the getAsciiStream() method if your application requires well-formed XML.

When XMLDescribeType=`blob`, your application should not use any of the methods for returning character data described in this section. In this case, the driver applies the standard JDBC character-to-binary conversion to the data, which returns the hexadecimal representation of the character data.

Binary data

When XMLDescribeType=blob, the driver returns XML data as binary data. The result set column is described with a column type of BLOB and the column type name is xml.

When XMLDescribeType=blob, your application can use the following methods to return XML data as binary data:

- ResultSet.getBytes()
- ResultSet.getBinaryStream()
- ResultSet.getBlob()
- ResultSet.getObject()
- CallableStatement.getBytes()
- CallableStatement.getBlob()
- CallableStatement.getObject()

The driver does not apply any data conversions to the XML data returned from the database server. These methods return a byte array or binary stream that contains the XML data encoded as UTF-8.

When XMLDescribeType=clob, your application should not use any of the methods for returning binary data described in this section. In this case, the driver applies the standard JDBC binary-to-character conversion to the data, which returns the hexadecimal representation of the binary data.

Inserting/updating XML data

The driver can insert or update XML data as character or binary data regardless of the setting of the XMLDescribeType connection property.

Character data

Your application can use the following methods to insert or update XML data as character data:

- PreparedStatement.setString()
- PreparedStatement.setCharacterStream()
- PreparedStatement.setClob()
- PreparedStatement.setObject()
- ResultSet.updateString()
- ResultSet.updateCharacterStream()
- ResultSet.updateClob()
- ReultSet.updateObject()

The driver converts the character representation of the data to the XML character set used by the database server and sends the converted XML data to the server. The driver does not parse or remove any XML processing instructions.

Your application can update XML data as ASCII data using the following methods:

- PreparedStatement.setAsciiStream()
- ResultSet.updateAsciiStream()

The driver interprets the data supplied to these methods using the ISO-8859-1 (latin 1) encoding. The driver converts the data from ISO-8859-1 to the XML character set used by the database server and sends the converted XML data to the server.

Binary data

Your application can use the following methods to insert or update XML data as binary data:

- PreparedStatement.setBytes()
- PreparedStatement.setBinaryStream()
- PreparedStatement.setBlob()
- PreparedStatement.setObject()
- ResultSet.updateBytes()
- ResultSet.updateBinaryStream()
- ResultSet.updateBlob()
- ReultSet.updateObject()

The driver does not apply any data conversions when sending XML data to the database server.

Support for Db2 pureScale

IBM introduced Db2 pureScale to provide scaleout active-active services for IBM Db2 running on AIX on Power Systems servers. It's designed to deliver distributed availability and scalability in a clustered database system. Db2 pureScale allows a single physical Db2 database to be accessed by concurrent instances of Db2 running across cluster members.

A Db2 pureScale shared disk cluster is composed of a group of independent servers, or members, that cooperate as a single system. A cluster architecture such as this provides applications access to more computing power when needed, while allowing computing resources to be used for other applications when database resources are not as heavily required. For example, in the event of a sudden increase in network traffic, a Db2 pureScale cluster can distribute the load over many nodes, a feature referred to as *transaction level workload balancing*. Db2 pureScale features are available to you simply by connecting to a Db2 pureScale system with the Db2 driver. No additional configuration is required.

Connection failover and client load balancing can be used in conjunction with a Db2 pureScale shared disk cluster.

Non-default schemas for catalog methods

To ensure that catalog methods function correctly when the CatalogSchema property is set to a schema other than the default schema, views for the catalog tables listed in the following table must exist in the specified schema. The views that are required depend on your Db2 database.

Table 10: Catalog Tables for Db2

Database	Catalog Tables
Db2 for Linux/UNIX/Windows	SYSCAT.TABLES SYSCAT.COLUMNS SYSCAT.PROCEDURES SYSCAT.PROCPARAMS SYSCAT.COLAUTH SYSCAT.TABAUTH SYSCAT.KEYCOLUSE SYSCAT.INDEXES SYSCAT.INDEXCOLUSE SYSCAT.REFERENCES SYSCAT.SYSSCHEMATA SYSCAT.TYPEMAPPINGS SYSCAT.DBAUTH
Db2 for z/OS	SYSIBM.SYSTABCONST SYSIBM.SYSTABLES SYSIBM.SYSSYNONYMS SYSIBM.SYSCOLUMNS SYSIBM.SYSPROCEDURES SYSIBM.SYSROUTINES SYSIBM.SYSPARMS SYSIBM.SYSCOLAUTH SYSIBM.SYSTABAUTH SYSIBM.SYSKEYS SYSIBM.SYSINDEXES SYSIBM.SYSRELS SYSIBM.SYSFOREIGNKEYS SYSIBM.SYSSCHEMAAUTH SYSIBM.SYSDBAUTH
Db2 for i	QSYS2.SYSCST QSYS2.SYSKEYCST QSYS2.SYSPROCS

Database	Catalog Tables
	QSYS2.SYSPARMS QSYS2.SYSTABLES QSYS2.SYSSYNONYMS QSYS2.SYSCOLUMNS QSYS2.SQLTABLEPRIVILEGES QSYS2.SYSKEYS QSYS2.SYSINDEXES QSYS2.SYSREFCST

Choosing a Db2 optimization class

By default, the driver requests the default optimization class. Most SQL statements are adequately optimized using the default optimization class, but you may want to use a different optimization class to improve performance. To specify an optimization class, set the CurrentQueryOptimization property.

In general, the lower optimization classes are better for simple or short-running SQL statements. The higher optimization classes consider more alternative query plans, which may benefit complex or long-running SQL statements, but they can incur significantly more compilation time, particularly if the SQL statement accesses multiple database tables. Benchmarks can help you determine if a better query plan has been generated for a SQL statement.

Consider the following general guidelines when choosing an optimization class:

- Start by using the default optimization class (class 5).
- If you want to try another optimization class other than the default, try class 1, 2, or 3 first.
- Use optimization class 1 or 2 if you have multiple tables with multiple join predicates on the same column, and if compilation time is a concern.
- Use a lower optimization class (0 or 1) for SQL statements that have short run times. These SQL statements often have the following characteristics:
  - Access a single table or only a few tables
  - Fetch a single row or only a few rows
  - Use fully qualified and unique indexes
- Use a higher optimization class (3, 5, or 7) for SQL statements that have longer run times of more than 30 seconds.
- Complex SQL statements may require different amounts of optimization to select the optimal query plan. Consider using higher optimization classes for SQL statements that have the following characteristics:
  - Access large tables
  - Contain multiple nested queries or joins
  - Contain multiple set operators (UNION, for example)

- Return multiple rows that match the search criteria
- Contain GROUP BY and HAVING clauses
- Contain nested table expressions

- Use class 9 only if you have extraordinary optimization requirements for a SQL statement.

Reauthentication

The Db2 driver supports reauthentication for Db2 for Linux/UNIX/Windows. The user performing the switch must have been granted the database SETSESSIONUSER permission and the SYSADM permission.

Note: Before performing reauthentication, applications must ensure that any statements or result sets created for one user are closed before switching the connection to another user.

Use the setCurrentUser() method in the ExtConnection interface to switch a user on a connection. The following table describes the driver-specific options supported by the setCurrentUser() method.

Table 11: ExtConnection.setCurrentUser() Options

Option	Description
CURRENT_SCHEMA	Specifies the name of the current schema. The value must be a valid Db2 schema name. If the setCurrentUser() method is called and this option is not specified or the value is set to #USER#, the schema is switched to the schema of the current user. If the setCurrentUser() method is called and this option is specified as an empty string, only the user is switched; the schema is not switched.
CURRENT_PATH	Specifies the current path for the database to use when locating stored procedures and functions. The value must be a valid path name for the Db2 CURRENT PATH special register. If the setCurrentUser() method is called and this option is not specified or the value is set to #USER#, the path is switched to the path of the current user. If the setCurrentUser() method is called and this option is specified as an empty string, only the user is switched; the path is not switched.

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

Refer to "JDBC Extensions" in the *Progress DataDirect for JDBC Drivers Reference* for more information about the setCurrentUser() method.

Db2 Workload Manager (WLM)

The Workload Manager (WLM) is a priority and resource manager within Db2 for Linux/UNIX/Windows. On z/OS, the WLM is part of the operating system. WLM prioritizes and matches Db2 workloads with available resources. *Db2 workloads* allow you to categorize similar types of work. For example, a database administrator can create a Db2 workload named Sales to service all connections that come from Sales applications.

The WLM automatically adjusts server resources, such as CPU and memory, based on the service class associated with a Db2 workload. Therefore, an application's performance is tied to the Db2 workload it is assigned to and, ultimately, to the service class associated with that workload. For example, an application that performs batch work nightly when resource usage is low can use the default workload. In contrast, sales updates that need to be processed quickly twice a day need to use a workload that is governed by a high priority service class.

It is important to understand that, unless specified otherwise, all work will run in the default workload that is governed by the default service class. To ensure the best performance, consult with your database administrator to verify that your application is associated with the appropriate Db2 workload and service class.

In addition to workload management, WLM also provides monitoring functionality that is useful for troubleshooting. For example, the database administrator can set threshold limits to detect long-running queries and gather information about those queries.

The Db2 driver allows your application to set client information in the Db2 database that can be used by the WLM to classify work. If you know that your database environment uses WLM, coordinate with your database administrator to determine which attributes you need to set. See the following topics for supported attributes.

- WLM attributes for Db2 for Linux/UNIX/Windows
- WLM attributes for Db2 for z/OS

WLM attributes for Db2 for Linux/UNIX/Windows

The following table lists the WLM attributes for Db2 for Linux/UNIX/Windows that map to information set by driver properties. Refer to your Db2 documentation for information about using these WLM attributes.

Table 12: WLM attributes for Db2 for Linux/UNIX/Windows

WLM Attribute	Driver Property	Description
APPLNAME	ApplicationName	Name of the application currently using the connection
CURRENT CLIENT_ACCTNG	AccountingInfo	Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID
CLIENT_PRDID	ProgramID	Product name and version of the driver on the client
CURRENT CLIENT_USERID	ClientUser	User ID for whom the application using the connection is performing work.
CURRENT CLIENT_WRKSTNNAME	ClientHostName	Host name of the client on which the application using the connection is running

WLM attributes for Db2 for z/OS

The following table lists the WLM attributes for Db2 for z/OS that map to information set by driver properties. Refer to your Db2 documentation for information about using these WLM attributes.

Table 13: WLM attributes for Db2 for z/OS

WLM Attribute	Driver Property	Description
Accounting Info (AI)	AccountingInfo	Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID
Correlation Info (CI)	ProgramID	Product name and version of the driver on the client
Collection Name (CN)	PackageCollection	Name of the collection or library (group of packages) to which Db2 packages are bound
Process Name (PC)	ApplicationName	Name of the application currently using the connection
Userid (UI)	ClientUser	User ID for whom the application using the connection is performing work.

Isolation levels

The Db2 driver supports the isolation levels listed in the following table. JDBC isolation levels are mapped to the appropriate Db2 transaction isolation levels as shown. The default isolation level is Read Committed.

Table 14: Supported Isolation Levels

JDBC Isolation Level	DB2 Isolation Level
None	No Commit [20]
Read Committed (default)	Cursor Stability
Read UnCommitted	Uncommitted Read
Repeatable Read	Read Stability
Serializable	Repeatable Read

Using scrollable cursors

The Db2 driver supports scroll-insensitive result sets and updatable result sets.

[20] Supported for Db2 for i versions that do not enable journaling.

Note: When the Db2 driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more SQLWarnings with detailed information.

Db2 for Linux, UNIX, Windows stored procedure cursor type OUT parameters

The Db2 driver supports cursor type OUT parameters for Db2 for Linux, UNIX, Windows stored procedures. To retrieve data from cursor output parameters, take the following steps:

1. Define a ResultSet object for each OUT parameter with the cursor data type.
2. Invoke the Connection.prepareCall method with the CALL statement as its argument to create a CallableStatement object.
3. Invoke the CallableStatement.registerOutParameter method to register the data types of parameters that are defined as OUT in the CREATE PROCEDURE statement. The driver data type for cursor type output parameters is com.ddtek.jdbc.extensions.ExtTypes.CURSOR.
4. Call the stored procedure.
5. Invoke the CallableStatement.getObject method to retrieve the ResultSet for each OUT cursor parameter. Calling CallableStatement.getString returns a name that is associated with the result set that is returned for the parameter. You can call only CallableStatement.getObject or CallableStatement.getString on a cursor parameter.
6. Retrieve rows from the ResultSet object for each OUT cursor parameter.
7. Close the ResultSet.

JTA support

To use JDBC distributed transactions through JTA with the Db2 driver, you must use one of the following databases:

- Db2 V8.*x* and higher for Linux/UNIX/Windows
- Db2 V9.1 for z/OS
- Db2 for i

Large object (LOB) support

Retrieving and updating Blobs, Clobs, and DBClobs is supported by the Db2 driver with the following databases:

- Db2 V8.*x* and higher for Linux/UNIX/Windows
- Db2 for z/OS
- Db2 for i

The Db2 driver supports Clobs up to a maximum of 2 GB with these databases. The Db2 driver supports retrieving and updating Clobs up to a maximum of 32 KB with all other supported Db2 databases.

Batch Inserts and Updates

The Db2 driver uses the native Db2 batch mechanism. By default, the methods used to set the parameter values of a batch performed using a PreparedStatement must match the database data type of the column with which the parameter is associated.

Db2 servers do not perform implicit data conversions, so specifying parameter values that do not match the column data type causes the Db2 server to generate an error. For example, to set the value of a Blob parameter using a stream or byte array when the length of the stream or array is less than 32 KB, you must use the setObject() method and specify the target JDBC type as BLOB; you cannot use the setBinaryStream() or setBytes() methods.

To remove the method-type restriction, set the BatchPerformanceWorkaround property to `true`. For example, you can use the setBinaryStream() or setBytes() methods to set the value of a Blob parameter regardless of the length of the stream or array; however, the parameter sets may not be executed in the order they were specified. Performance may be decreased because the driver must convert the parameter data to the correct data type and re-execute the statement.

Parameter metadata support

The Db2 driver supports returning parameter metadata as described in this section.

Insert and Update statements

The Db2 driver supports returning parameter metadata for all types of SQL statements with the following databases:

- DB2 V8.*x* and higher for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 for i

Select statements

The Db2 driver supports returning parameter metadata for all types of SQL statements with the following Db2 databases:

- DB2 V8.*x* and higher for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 for i

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

  ```
  SELECT * FROM foo WHERE bar > ?
  ```

  In this case, the value expression "bar" can be targeted against the table "foo" to determine the appropriate metadata for the parameter.

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

  ```
  SELECT * FROM foo WHERE (SELECT x FROM y WHERE z = 1) < ?
  ```

The following Select statements show further examples for which parameter metadata can be returned:

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2 WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

ANSI SQL 92 entry-level predicates in a WHERE clause containing GROUP BY, HAVING, or ORDER BY statements are supported. For example:

```
SELECT * FROM t1 WHERE col = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM t1,t2 WHERE t1.col1 = ?
```

Fully qualified names and aliases are supported. For example:

```
SELECT a, b, c, d FROM T1 AS A, T2 AS B WHERE A.a = ? and B.b = ?"
```

Stored procedures

The Db2 driver supports returning parameter metadata for stored procedure arguments.

ResultSet metadata support

If your application requires table name information, the Db2 driver can return table name information in ResultSet metadata for Select statements. If you set the ResultSetMetaDataOptions property to 1, the Db2 driver performs additional processing to determine the correct table name for each column in the result set when the ResultSetMetaData.getTableName() method is called. Otherwise, the getTableName() method may return an empty string for each column in the result set.

The table name information that is returned by the Db2 driver depends on whether the column in a result set maps to a column in a table in the database. For each column in a result set that maps to a column in a table in the database, the Db2 driver returns the table name associated with that column. For columns in a result set that do not map to a column in a table (for example, aggregates and literals), the Db2 driver returns an empty string.

The Select statements for which ResultSet metadata is returned may contain aliases, joins, and fully qualified names. The following queries are examples of Select statements for which the ResultSetMetaData.getTableName() method returns the correct table name for columns in the Select list:

```
SELECT id, name FROM Employee
SELECT E.id, E.name FROM Employee E
SELECT E.id, E.name AS EmployeeName FROM Employee E
```

```
SELECT E.id, E.name, I.location, I.phone FROM Employee E, EmployeeInfo I
   WHERE E.id = I.id
SELECT id, name, location, phone FROM Employee, EmployeeInfo WHERE id = empId
SELECT Employee.id, Employee.name, EmployeeInfo.location, EmployeeInfo.phone
   FROM Employee, EmployeeInfo WHERE Employee.id = EmployeeInfo.id
```

The table name returned by the driver for generated columns is an empty string. The following query is an example of a Select statement that returns a result set that contains a generated column (the column named "upper").

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)} AS upper FROM Employee E
```

The Db2 driver also can return schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information. For example, for the following statement, the Db2 driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

The additional processing required to return table name, schema name, and catalog name information is only performed if the ResultSetMetaData.getTableName(), ResultSetMetaData.getSchemaName(), or ResultSetMetaData.getCatalogName() methods are called.

Rowset support

The Db2 driver supports any JSR 114 implementation of the RowSet interface, including:

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See https://www.jcp.org/en/jsr/detail?id=114 for more information about JSR 114.

Auto-generated keys support

The Db2 driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the Db2 driver is the value of an auto-increment column.

An application can return values of auto-generated keys when it executes an Insert statement. How you return these values depends on whether you are using an Insert statement with a Statement object or with a PreparedStatement object, as outlined in the following scenarios:

- When using an Insert statement with a Statement object, the driver supports the following form of the Statement.execute and Statement.executeUpdate methods to instruct the driver to return values of auto-generated keys:
  - `Statement.execute(String sql, int autoGeneratedKeys)`
  - `Statement.execute(String sql, int[] columnIndexes)`

- `Statement.execute(String sql, String[] columnNames)`
- `Statement.executeUpdate(String sql, int autoGeneratedKeys)`
- `Statement.executeUpdate(String sql, int[] columnIndexes)`
- `Statement.executeUpdate(String sql, String[] columnNames)`

- When using an Insert statement with a PreparedStatement object, the driver supports the following form of the Connection.prepareStatement method to instruct the driver to return values of auto-generated keys:
  - `Connection.prepareStatement(String sql, int autoGeneratedKeys)`
  - `Connection.prepareStatement(String sql, int[] columnIndexes)`
  - `Connection.prepareStatement(String sql, String[] columnNames)`

An application can retrieve values of auto-generated keys using the Statement.getGeneratedKeys method. This method returns a ResultSet object with a column for each auto-generated key.

Refer to "Designing JDBC applications for performance optimization" in the *Progress DataDirect for JDBC Drivers Reference* for information about how auto-generated keys can improve performance.

TCP KeepAlive support

The Db2 driver supports the KeepAlive connection setting, which allows the TCP socket to keep an idle connection active by transmitting packets between the client and server. See "KeepAlive" for more details.

Note: The process to configure the KeepAlive property described in this section is valid at the time of publication. For latest information on the process, refer to the following links for your operating system:

- Windows
- Linux
- Solaris
- HP-UX
- AIX

The following steps describe how you can enable this feature for the following operating systems.

Windows:

1. Launch the registry and navigate to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters.
2. Create a `KeepAliveInterval` parameter of the REG_DWORD type with an interval value in milliseconds. For example, 10000 for 10 seconds.
3. Create a `KeepAliveTime` parameter of the REG_DWORD type with the number of milliseconds to wait before resending a KeepAlive packet. For example, 3600000 for 1 hour.
4. Restart the operating system.

Linux:

1. Set the time interval between the probes using the following command from the root account:

   ```
   echo interval_value > /proc/sys/net/ipv4/tcp_keepalive_intvl
   ```

2. Set the maximum number of probes using the following command:

   ```
   echo probe_value > /proc/sys/net/ipv4/tcp_keepalive_probes
   ```

3. Restart the operating system.

Solaris:

1. Set the KeepAlive interval value using the following command from the root account:

   ```
   ndd -set /dev/tcp tcp_keepalive_interval interval_value
   ```

2. Restart the operating system.

HP-UX:

1. Set the KeepAlive interval value using the following command from the root account:

   ```
   ndd -set /dev/tcp tcp_keepalive_interval interval_value
   ```

2. Set the maximum number of probes using the following command:

   ```
   ndd -set /dev/tcp tcp_keepalive_kill probe_value
   ```

3. Restart the operating system.

AIX:

1. Set the KeepAlive interval value using the following command from the root account:

   ```
   no -o tcp_keepidle=interval_value
   ```

2. Set the interval between KeepAlive packets to maintain a connection using the following command:

   ```
   no -o tcp_keepintvl=interval_value
   ```

3. Restart the operating system.

Note: To permanently change the values for AIX, HP-UX, Solaris, and Linux, set the values while the system is starting by modifying the startup script.

See also

KeepAlive on page 138

5

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.

- TLS/SSL Encryption
- Bulk load properties
- Failover properties
- Timeout properties
- Client information properties
- Statement pooling properties
- Additional properties

General properties

The following table summarizes connection properties required to connect to a database.

Property	Data Source Method	Default
Database on page 125	`setDatabase(String)` `getDatabase()`	No default value
LocationName on page 143	`setLocationName(String)` `getLocationName()`	No default value
PortNumber on page 150	`setPortNumber(Int)` `getPortNumber()`	`50000`
ServerName on page 160	`setServerName(String)` `getServerName()`	No default value

User ID and password authentication properties

The following table summarizes the connection properties required for user ID and password authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 111	`setAuthenticationMethod(String)` `getAuthenticationMethod()`	`clearText`
Password on page 150	`setPassword(String)` `getPassword()`	No default value
User on page 166	`setUser(String)` `getUser()`	No default value

User ID and password authentication with AES encryption properties

The following table summarizes the connection properties required for user ID and password authentication using AES encryption with random number generator for secure seeding.

Property	Data Source Method	Default
AuthenticationMethod on page 111	`setAuthenticationMethod(String)` `getAuthenticationMethod()`	`clearText`
RandomGenerator on page 155	`setRandomGenerator(String)` `getRandomGenerator()`	`secureRandom`
SecureRandomAlgorithm on page 158	`setSecureRandomAlgorithm(String)` `getSecureRandomAlgorithm()`	No default value
Password on page 150	`setPassword(String)` `getPassword()`	No default value
User on page 166	`setUser(String)` `getUser()`	No default value

GSS plug-in authentication properties

The following table summarizes the connection properties required for GSS plug-in authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 111	`setAuthenticationMethod(String)` `getAuthenticationMethod()`	`clearText`
GSSPluginName on page 132	`setGSSPluginName(String)` `getGSSPluginName()`	No default value
GSSPluginObject on page 133	`setGSSPluginObject()` `getGSSPluginObject()`	No default value
Password on page 150	`setPassword(String)` `getPassword()`	No default value
User on page 166	`setUser(String)` `getUser()`	No default value

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 152	`setProxyHost(String)` `getProxyHost()`	No default value
ProxyPassword on page 152	`setProxyPassword(String)` `getProxyPassword()`	No default value
ProxyPort on page 153	`setProxyPort(Int)` `getProxyPort()`	`0`
ProxyUser on page 154	`setProxyUser(String)` `getProxyUser()`	No default value

Db2 package properties

The following table summarizes the connection properties required for creating Db2 packages.

Property	Data Source Method	Default
CreateDefaultPackage on page 122	`setCreateDefaultPackage(Boolean)` `getCreateDefaultPackage()`	`false`
DynamicSections on page 126	`setDynamicSections(Int)` `getDynamicSections()`	`200` (prepared statements)
PackageCollection on page 148	`setPackageCollection(String)` `getPackageCollection()`	`NULLID`
PackageOwner on page 149	`setPackageOwner(String)` `getPackageOwner()`	`NULL`
ReplacePackage on page 157	`setReplacePackage(Boolean)` `getReplacePackage()`	`false`

TLS/SSL encryption properties

The following table summarizes the connection properties required for TLS/SSL encryption.

Property	Data Source Method	Default
CryptoProtocolVersion on page 123	`setCryptoProtocolVersion(String)` `getCryptoProtocolVersion()`	`TLSv1.3`

Property	Data Source Method	Default
EncryptionMethod on page 127	`setEncryptionMethod(String)` `getEncryptionMethod()`	`noEncryption`
HostNameInCertificate on page 134	`setHostNameInCertificate(String)` `getHostNameInCertificate()`	No default value
KeyPassword on page 139	`setKeyPassword(String)` `getKeyPassword()`	No default value
KeyStore on page 140	`setKeyStore(String)` `getKeyStore()`	No default value
KeyStorePassword on page 141	`setKeyStorePassword(String)` `getKeyStorePassword()`	No default value
TrustStore on page 164	`setTrustStore(String)` `getTrustStore()`	No default value
TrustStorePassword on page 165	`setTrustStorePassword(String)` `getTrustStorePassword()`	No default value
ValidateServerCertificate on page 167	`setValidateServerCertificate(Boolean)` `getValidateServerCertificate()`	`true`

Bulk load properties

The following table contains the only connection property that affects how bulk load works with the driver.

Property	Data Source Method	Default
BulkLoadBatchSize on page 113	`setBulkLoadBatchSize(Long)` `getBulkLoadBatchSize()`	`2048`

Failover properties

The following table summarizes the connection properties used for configuring failover.

Property	Data Source Method	Default
AlternateID on page 108	`setAlternateID(String)` `getAlternateID()`	No default value

Property	Data Source Method	Default
AlternateServers on page 109	`setAlternateServers(String)` `getAlternateServers()`	No default value
ConnectionRetryCount on page 119	`setConnectionRetryCount(Int)` `getConnectionRetryCount()`	`5`
ConnectionRetryDelay on page 120	`setConnectionRetryDelay(Int)` `getConnectionRetryDelay()`	`1` (second)
FailoverGranularity on page 128	`setFailoverGranularity(String)` `getFailoverGranularity()`	`nonAtomic`
FailoverMode on page 129	`setFailoverMode(String)` `getFailoverMode()`	`connect`
FailoverPreconnect on page 130	`setFailoverPreconnect(Boolean)` `getFailoverPreconnect()`	`false`
LoadBalancing on page 142	`setLoadBalancing(Boolean)` `getLoadBalancing()`	`false`

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
LoginTimeout on page 144	`setLoginTimeout(Int)` `getLoginTimeout()`	`0`
QueryTimeout on page 154	`setQueryTimeout(Int)` `getQueryTimeout()`	`0`

Client information properties

The following table summarizes connection properties that can be used to return client information.

Property	Data Source Method	Default
AccountingInfo on page 106	`setAccountingInfo(String)` `getAccountingInfo()`	No default value

Property	Data Source Method	Default
ApplicationName on page 110	`setApplicationName(String)` `getApplicationName()`	No default value
ClientHostName on page 116	`setClientHostName(String)` `getClientHostName()`	No default value
ClientUser on page 117	`setClientUser(String)` `getClientUser()`	No default value
ProgramID on page 151	`setProgramID(String)` `getProgramID()`	No default value

Statement pooling properties

The following table summarizes statement pooling connection properties.

Property	Data Source Method	Default
ImportStatementPool on page 135	`setImportStatementPool(String)` `getImportStatementPool()`	No default value
MaxPooledStatements on page 146	`setMaxPooledStatements(Int)` `getMaxPooledStatements()`	`0`
RegisterStatementPoolMonitorMBean on page 156	`setRegisterStatementPoolMonitor MBean(Boolean)` `getRegisterStatementPoolMonitor MBean()`	`false`

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
AddToCreateTable on page 107	`setAddToCreateTable(String)` `getAddToCreateTable()`	No default value
AllowImplicitResultSetCloseForXA on page 108	`setAllowImplicitResultSetClose ForXA(Boolean)` `getAllowImplicitResultSetClose ForXA()`	`false`
CatalogOptions on page 114	`setCatalogOptions(Int)` `getCatalogOptions()`	`2`

Property	Data Source Method	Default
CatalogSchema on page 115	`setCatalogSchema(String)` `getCatalogSchema()`	`SYSCAT` (Db2 for Linux/UNIX/Windows) `SYSIBM` (Db2 for z/OS) `QSYS2` (Db2 for i)
CharsetFor65535 on page 116	`setCharsetFor65535(String)` `getCharsetFor65535()`	No default value
CodePageOverride on page 118	`setCodePageOverride(String)` `getCodePageOverride()`	No default value
ConcurrentAccessResolution on page 119	`setConcurrentAccessResolution (String)` `getConcurrentAccessResolution()`	`auto`
ConvertNull on page 121	`setConvertNull(Int)` `getConvertNull()`	`1`
CurrentFunctionPath on page 124	`setCurrentFunctionPath(String)` `getCurrentFunctionPath()`	`null`
CurrentQueryOptimization on page 124	`setCurrentQueryOptimization(Int)` `getCurrentQueryOptimization()`	`-1`
EnableCancelTimeout on page 127	`setEnableCancelTimeout(Boolean)` `getEnableCancelTimeout()`	`false`
Grantee on page 131	`setGrantee(String)` `getGrantee()`	`PUBLIC`
GrantExecute on page 131	`setGrantExecute(Boolean)` `getGrantExecute()`	`true`
InitializationString on page 135	`setInitializationString(String)` `getInitializationString()`	No default value
InsensitiveResultSetBufferSize on page 136	`setInsensitiveResultSetBuffer Size(Int)` `getInsensitiveResultSetBufferSize()`	`2048`
JavaDoubleToString on page 137	`setJavaDoubleToString(Boolean)` `getJavaDoubleToString()`	`false`

Property	Data Source Method	Default
JDBCBehavior on page 138	`setJDBCBehavior(Int)` `getJDBCBehavior()`	`1`
KeepAlive on page 138	`setKeepAlive(Boolean)` `getKeepAlive()`	`false`
LobStreamingProtocol on page 142	`setLobStreamingProtocol(String)` `getLobStreamingProtocol()`	`streaming`
LongDataCacheSize on page 145	`setLongDataCacheSize(Int)` `getLongDataCacheSize()`	`2048`
OptimizationProfile on page 147	`setOptimizationProfile(String)` `getOptimizationProfile()`	`null`
OptimizationProfileToFlush on page 147	`setOptimizationProfileToFlush (String)` `getOptimizationProfileToFlush()`	`null`
ResultSetMetaDataOptions on page 157	`setResultSetMetaDataOptions(Int)` `getResultSetMetaDataOptions()`	`0`
SendStreamAsBlob on page 159	`setSendStreamAsBlob(Boolean)` `getSendStreamAsBlob()`	`false`
StripNewlines on page 163	`setStripNewlines(Boolean)` `getStripNewlines()`	`false`
SpyAttributes on page 161	`setSpyAttributes(String)` `getSpyAttributes()`	No default value
UseCurrentSchema on page 165	`setUseCurrentSchema(Boolean)` `getUseCurrentSchema()`	`false`

Property	Data Source Method	Default
WithHoldCursors on page 167	`setWithHoldCursors(Boolean)` `getWithHoldCursors()`	`true`
XMLDescribeType on page 168	`setXMLDescribeType(String)` `getXMLDescribeType()`	No default value

For details, see the following topics:

- AccountingInfo
- AddToCreateTable
- AllowImplicitResultSetCloseForXA
- AlternateID
- AlternateServers
- ApplicationName
- AuthenticationMethod
- BatchPerformanceWorkaround
- BulkLoadBatchSize
- CatalogOptions
- CatalogSchema
- CharsetFor65535
- ClientHostName
- ClientUser
- CodePageOverride
- ConcurrentAccessResolution
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CreateDefaultPackage
- CryptoProtocolVersion
- CurrentFunctionPath
- CurrentQueryOptimization
- Database
- DynamicSections

- EnableCancelTimeout
- EncryptionMethod
- FailoverGranularity
- FailoverMode
- FailoverPreconnect
- Grantee
- GrantExecute
- GSSPluginName
- GSSPluginObject
- HostNameInCertificate
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- JDBCBehavior
- KeepAlive
- KeyPassword
- KeyStore
- KeyStorePassword
- LoadBalancing
- LobStreamingProtocol
- LocationName
- LoginTimeout
- LongDataCacheSize
- MaxPooledStatements
- OptimizationProfile
- OptimizationProfileToFlush
- PackageCollection
- PackageOwner
- Password
- PortNumber
- ProgramID
- ProxyHost

- ProxyPassword
- ProxyPort
- ProxyUser
- QueryTimeout
- RandomGenerator
- RegisterStatementPoolMonitorMBean
- ReplacePackage
- ResultSetMetaDataOptions
- SecureRandomAlgorithm
- SendStreamAsBlob
- ServerName
- SpyAttributes
- StripNewlines
- TrustStore
- TrustStorePassword
- UseCurrentSchema
- User
- ValidateServerCertificate
- WithHoldCursors
- XMLDescribeType

AccountingInfo

Purpose

Specifies accounting information to be stored in the database. This property sets the CURRENT CLIENT_ACCTNG register (Db2 for Linux/UNIX/Windows) or the CLIENT ACCTNG register (Db2 for z/OS and Db2 for i) in the database. This value is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the accounting information.

Notes

- 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 getAccountingInfo()
```

```
public void setAccountingInfo(String)
```

Default

No default value

Data Type

String

See also

Client information on page 72

AddToCreateTable

Purpose

A string that is appended to the end of all CREATE statements. This property can be used to add an "in database" clause.

Valid Values

string

where:

string

is the set of characters appended to all CREATE statements.

Data Source Methods

```
public String getAddToCreateTable()
```

```
public void setAddToCreateTable(String)
```

Default

No default value

Data Type

String

AllowImplicitResultSetCloseForXA

Purpose

Specifies whether result sets in distributed transactions are automatically closed when all rows of the result sets have been returned.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver automatically closes result sets in distributed transactions when all rows of the result sets have been returned.

If set to `false`, the driver does not automatically close result sets in distributed transactions when all rows of the result sets have been returned.

Data Source Methods

```
public Boolean getAllowImplicitResultSetCloseForXA()
```

```
public void setAllowImplicitResultSetCloseForXA(Boolean)
```

Default

`true`

Data Type

Boolean

AlternateID

Purpose

Specifies the name of the schema to be used to qualify unqualified database objects in dynamically prepared SQL statements. This property sets the name of the schema in the DB2 CURRENT SCHEMA special register. If the attempt to change the schema fails, the connection fails and you receive the message `Invalid value for AlternateID`. Refer to your Db2 documentation for permission requirements imposed by the database.

Valid Values

`string`

where:

`string`

is a valid Db2 schema name.

Data Source Methods

```
public String getAlternateID()
```

```
public void setAlternateID(String)
```

Default

No default value

Data type

String

AlternateServers

Purpose

Specifies 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 [;...]]]...)
```

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 the port number of the primary server is unspecified, the default port number of 50000 is used.

Optional connection properties are DatabaseName (Db2 for Linux/UNIX/Windows) and LocationName (Db2 for z/OS and Db2 for i).

Example

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

```
jdbc:datadirect:db2://server1:50000;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,
server3:50000;DatabaseName=TEST3)
```

Notes

- If using failover with Db2 High Availability Disaster Recovery (HADR), the primary server and any alternate server must be the primary server and a standby server, respectively, that is configured in your HADR system.

Data Source Methods

```
public String getAlternateServers()
```

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

Default

No default value

Data type

String

See also

FailoverMode on page 129

Failover on page 70

ApplicationName

Purpose

Specifies the name of the application to be stored in the database. This property sets the CURRENT CLIENT_APPLNAME register (Db2 for Linux/UNIX/Windows) or CLIENT APPLNAME register (Db2 for z/OS and Db2 for i) in the database. For Db2 for Linux/UNIX/Windows, it also sets the APPL_NAME column of the SYSIBMADM.APPLICATIONS table. These values are used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the name of the application.

Notes

- 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

No default value

Data Type

String

See also

Client information on page 72

AuthenticationMethod

Purpose

Determines which authentication method the driver uses when it establishes a connection.

When user ID/password authentication is used, the encryption method that is used for user IDs and passwords is negotiated during the connection process. Supported encryption methods are:

- Advanced Encryption Standard (AES)
- Data Encryption Standard (DES)

To use AES encryption, the following requirements and restrictions apply:

- AES is supported for the following Db2 databases:
  - Db2 for Linux/UNIX/Windows
  - Db2 for z/OS
- The Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files must be installed on the client or application server. You can obtain these files from the following URL:

  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- The Db2 authentication parameter on the database server must be set to a value of `SERVER_ENCRYPT`.
- For Db2 for Linux/UNIX/Windows, the Db2 alternate_auth_enc parameter on the database server must be set to allow AES encryption.
- AES encryption cannot be used if the EncryptionMethod property is set to a value of `DBEncryption` or `requestDBEncryption`.

Valid Values

`clearText` | `client` | `encryptedPassword` | `encryptedPasswordAES` | `encryptedUIDPassword` | `encryptedUIDPasswordAES` | `kerberos` | `pluginSecurity`

Behavior

If set to `clearText`, the driver uses user ID/password authentication. The driver sends the user ID and password in clear text to the Db2 server for authentication. If a user ID and password are not specified, the driver throws an exception.

If set to `client`, the driver uses client authentication. The Db2 server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any user ID or password specified.

If set to `encryptedPassword`, the driver uses user ID/password authentication. The driver sends a user ID in clear text and an encrypted password to the Db2 server for authentication. If the requirements for AES encryption are met, the driver uses AES encryption; otherwise, the driver allows a downgrade to DES encryption. If the EncryptionMethod property is set to a value of `DBEncryption` or `requestDBEncryption`, the driver downgrades encryption to DES. If a user ID and password are not specified, the driver throws an exception.

If set to `encryptedPasswordAES`, the driver uses user ID/password authentication. The driver sends a clear text user ID and an AES-encrypted password to the Db2 server for authentication. The driver throws an exception in the following cases:

- If the database server indicates encryption must be downgraded to DES

- If a user ID and password are not specified
- If the EncryptionMethod property is set to a value of `DBEncryption` or `requestDBEncryption`

If set to `encryptedUIDPassword`, the driver uses user ID/password authentication. The driver sends an encrypted user ID and password to the Db2 server for authentication. If the requirements for AES encryption are met, the driver uses AES encryption; otherwise, the driver allows a downgrade to DES encryption. If the EncryptionMethod property is set to a value of `DBEncryption` or `requestDBEncryption`, the driver downgrades encryption to DES. If a user ID and password are not specified, the driver throws an exception.

If set to `encryptedUIDPasswordAES`, the driver uses user ID/password authentication. The driver sends an AES-encrypted user ID and password to the Db2 server for authentication. The driver throws an exception in the following situations:

- If the database server indicates encryption must be downgraded to DES
- If a user ID and password are not specified
- If the EncryptionMethod property is set to a value of `DBEncryption` or `requestDBEncryption`.

If set to `kerberos`, the driver uses Kerberos authentication. The driver ignores any user ID or password specified.

If set to `pluginSecurity`, the driver uses security plug-ins for authentication. The driver supports GSS plug-in for authentication and it is configured using the GSSPluginName and GSSPluginObject properties.

Notes

- The User property provides the user ID. The Password property provides the password. The EncryptionMethod property determines whether the driver uses data encryption.
- If you enable AES encryption by setting the AuthenticationMethod property to `encryptedPasswordAES` or `encryptedUIDPasswordAES`, a random number generator is used for secure seeding. Secure seeding is configurable through the RandomGenerator and SecureRandomAlgorithm connection properties.

Data Source Methods

```
public String getAuthenticationMethod()
```

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

Default

`clearText`

Data Type

String

See also

- Authentication on page 53
- Random number generator secure seeding on page 55
- RandomGenerator on page 155
- SecureRandomAlgorithm on page 158
- Kerberos authentication on page 56
- GSS plug-in authentication on page 62
- Data encryption on page 64

BatchPerformanceWorkaround

Purpose

The driver uses the native Db2 batch mechanism. This property determines whether restrictions are enforced to facilitate data conversions.

Valid Values

`true` | `false`

Behavior

If set to `true`, restrictions are removed; however, parameter sets may not be executed in the order that they were specified.

If set to `false`, the methods that are used to set the parameter values of a batch operation that is performed using a PreparedStatement must match the database data type of the column with which the parameter is associated. This is because Db2 servers do not perform implicit data conversions.

Data Source Methods

```
public Boolean getBatchPerformanceWorkaround()
```

```
public void setBatchPerformanceWorkaround(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Batch Inserts and Updates on page 89

BulkLoadBatchSize

Purpose

Provides a suggestion to the driver for the number of rows to load to the database at a time when bulk loading data. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

Valid Values

x

where:

x

is a positive integer that represents a number of rows.

Data Source Methods

```
public Long getBulkLoadBatchSize()
public void setBulkLoadBatchSize(Long)
```

Default

`2048`

Data Type

Long

Notes

- This property suggests the number of rows regardless of which bulk load method is used: using a DDBulkLoad object or using bulk load for batch inserts.
- The DDBulkObject.setBatchSize() method overrides the value set by this property.

  Refer to "JDBC extensions" in the *Progress DataDirect for JDBC Drivers Reference* for more information about bulk load methods.

See also

Bulk Load on page 73

CatalogOptions

Purpose

Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods.

Valid Values

`0` | `2` | `6`

Behavior

If set to `0`, result sets do not contain synonyms or remarks.

If set to `2`, result sets contain synonyms and remarks that are returned from the following DatabaseMetaData methods: getColumns(), getExportedKeys(), getFunctionColumns(), getFunctions(), getImportedKeys(), getIndexInfo(), getPrimaryKeys(), getProcedureColumns(), and getProcedures().

If set to `6`, a hint is provided to the driver to emulate getColumns() calls using the ResultSetMetaData object instead of querying database catalogs for column information. Result sets contain synonyms, but not remarks. Using emulation can improve performance because the SQL statement that is formulated by the emulation is less complex than the SQL statement that is formulated using getColumns(). The argument to getColumns() must evaluate to a single table. If it does not, because of a wildcard or null value, for example, the driver reverts to the default behavior for getColumns() calls.

Data Source Methods

```
public Integer getCatalogOptions()
```

```
public void setCatalogOptions(Integer)
```

Default

2

Data Type

Int

See also

Performance considerations on page 73

CatalogSchema

Purpose

Specifies the Db2 schema to use for catalog methods. To improve performance, views of system catalog tables can be created in a schema other than the default catalog schema. Setting this property to a schema that contains views of the catalog tables allows the driver to use those views. To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your Db2 database. See "Non-default schemas for catalog methods" for views for catalog tables that must exist in the specified schema.

Valid Values

`string`

where:

`string`

is the name of a valid Db2 schema.

Data Source Methods

```
public String getCatalogSchema()
```

```
public void setCatalogSchema(String)
```

Default

`SYSCAT` (Db2 for Linux/UNIX/Windows) | `SYSIBM` (Db2 for z/OS) | `QSYS2` (Db2 for i)

Data Type

String

See also

Non-default schemas for catalog methods on page 82

Performance considerations on page 73

CharsetFor65535

Purpose

Specifies the code page to be used by the driver to convert character data that is stored as bit data in character columns (Char, Varchar, Longvarchar, Char for Bit Data, Varchar for Bit Data, Longvarchar for Bit Data) defined with CCSID 65535. All character data that is stored as bit data and returned from the database using columns defined with CCSID 65535 is converted using the specified code page. This property has no effect when writing data to character columns that are defined with CCSID 65535.

Valid Values

string

where:

`string`

is the name of a valid code page that is supported by your JVM.

Example

```
CP950
```

Data Source Methods

```
public String getCharsetFor65535()
```

```
public void setCharsetFor65535(String)
```

Default

No default value

Data Type

String

ClientHostName

Purpose

Specifies the host name of the client machine to be stored in the database. This property sets the CURRENT CLIENT_WRKSTNNAME register (Db2 for Linux/UNIX/Windows) or CLIENT WRKSTNNAME register (Db2 for z/OS and Db2 for i) in the database. This value is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the host name of the client machine.

Notes

- 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 getClientHostName()
```

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

Default

No default value

Data Type

String

See also

Client information on page 72

ClientUser

Purpose

Specifies the user ID to be stored in the database. This property sets the CURRENT CLIENT_USERID register (Db2 for Linux/UNIX/Windows) and CLIENT USERID register (Db2 for z/OS and Db2 for i) in the database. This value is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is a valid user ID.

Notes

- 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 getClientUser()
```

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

Default

No default value

Data Type

String

See also

Client information on page 72

CodePageOverride

Purpose

Specifies the code page to be used by the driver to convert Character and Clob data. The specified code page overrides the default database code page or column collation. All Character and Clob data that is returned from or written to the database is converted using the specified code page.

By default, the driver automatically determines which code page to use to convert Character data. Use this property only if you need to change the driver's default behavior.

Valid Values

string

where:

string

is the name of a valid code page that is supported by your JVM.

Example

```
CP950
```

Data Source Methods

```
public String getCodePageOverride()
```

```
public void setCodePageOverride(String)
```

Default

No default value

Data Type

String

ConcurrentAccessResolution

Description

Determines whether a read transaction can access committed rows that are locked by a write transaction when the application isolation level is Read Committed (Db2 Cursor Stability) or Repeatable Read (Db2 Read Stability).

This property only applies to connections to Db2 V9.7 for Linux/UNIX/Windows and higher databases.

Valid Values

`auto` | `useCurrentlyCommitted` | `waitForOutcome`

Behavior

If set to `auto`, the driver determines whether read transactions can access currently committed data when lock contention occurs by checking the setting of the Db2 cur_commit parameter on the database server. If the cur_commit parameter is set to `ON`, read transactions can access currently committed data.

If set to `useCurrentlyCommitted`, the driver allows read transactions to access currently committed data if the data is being updated or deleted. Read transactions skip rows that are being inserted.

If set to `waitForOutcome`, read transactions wait for a commit or rollback operation if they encounter data that is being updated or deleted. Read transactions do not skip rows that are being inserted.

Data Source Methods

```
public String getConcurrentAccessResolution()
```

```
public void setConcurrentAccessResolution(String)
```

Default

`auto`

Data Type

String

ConnectionRetryCount

Purpose

Specifies the number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers 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 database server to which it tried to connect.

Example

If this property is set to 2 and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) 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

5

Data Type

Int

See also

Failover on page 70

ConnectionRetryDelay

Purpose

Specifies 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`, this property is set to `3`, and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) 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

`1` (second)

Data Type

Int

See also

Failover on page 70

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`0` | `1`

Behavior

If set to `0`, 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.

If set to `1`, 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 the data type of the column value.

Data Source Methods

```
public Integer getConvertNull()
```

```
public void setConvertNull(Integer)
```

Default

1

Data Type

Int

CreateDefaultPackage

Purpose

Determines whether the driver automatically creates required Db2 packages.

For Db2 for Linux/UNIX/Windows, this property must be used in conjunction with the ReplacePackage property.

For Db2 for z/OS and Db2 for i, Db2 packages are created in the collection or library that is specified by the PackageCollection property.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver automatically creates required Db2 packages, even if they already exist. Existing Db2 packages are replaced by the new packages.

If set to `false`, the driver determines if the required Db2 packages exist. If they do not, the driver automatically creates them.

Data Source Methods

```
public Boolean getCreateDefaultPackage()
```

```
public void setCreateDefaultPackage(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Db2 packages on page 76

CryptoProtocolVersion

Purpose

Specifies a cryptographic protocol or comma-separated list of cryptographic protocols that can be used when TLS/SSL is enabled using the EncryptionMethod connection property.

Valid Values

cryptographic_protocol [[, *cryptographic_protocol*]...]

where:

cryptographic_protocol

is one of the following cryptographic protocols:

TLSv1.3 | TLSv1.2

Example

If your server supports TLSv1.3 and TLSv1.2, you can specify acceptable cryptographic protocols with the following key-value pair:

```
CryptoProtocolVersion=TLSv1.3,TLSv1.2
```

Notes

- The TLSv1.3 protocol works with Java SE 11 or higher by default.
- To enable the TLSv1.3 protocol when using Java SE 8, set the `jdk.tls.client.protocols` Java system property to `TLSv1.3`. For example, `$ java -Djdk.tls.client.protocols="TLSv1.3" myApp`.

  In the following versions of Oracle JDK and OpenJDK, support for the TLSv1.3 protocol is not enabled by default.

  - Oracle JDK 8u261 or later but earlier than Oracle JDK 8u341
  - OpenJDK 8u272 or later but earlier than OpenJDK 8u352

- When multiple protocols are specified, the driver uses the highest version supported by the server. If none of the specified protocols are supported by the server, the connection fails and the driver returns an error.
- When no value has been specified for CryptoProtocolVersion, the cryptographic protocol used depends on the highest protocol version supported by the server and the highest protocol version supported by the JDK. Refer to the database management system documentation for information on which cryptographic protocols are supported.

Data Source Methods

```
public String getCryptoProtocolVersion()
```

```
public void setCryptoProtocolVersion(String)
```

Default

TLSv1.3

Data Type

String

See also

- EncryptionMethod on page 127
- Data encryption on page 64

CurrentFunctionPath

Purpose

Specifies a list of Db2 schema names that are used to resolve unqualified function names and data type references in dynamically prepared SQL statements. It also is used to resolve unqualified stored procedure names that are specified in CALL statements. This property sets the CURRENT PATH register in the database.

Valid Values

schema_name[[,*schema_name*]...]

where:

schema_name

Is a valid Db2 schema name.

Data Source Methods

```
public String getCurrentFunctionPath()
```

```
public void setCurrentFunctionPath(String)
```

Default

`null`

Data Type

String

CurrentQueryOptimization

Purpose

Specifies the Db2 optimization class that is performed by the database when generating a query plan for a SQL statement.

This property only applies to connections to Db2 V9.7 and higher for Linux/UNIX/Windows.

Valid Values

```
-1 | 0 | 1 | 2 | 3 | 5 | 7 | 9
```

Behavior

If set to `-1`, the default class configured for the database server is used.

If set to `0`, a minimum amount of optimization is performed. This class is useful for simple dynamic SQL to well-indexed database tables.

If set to `1`, optimization that is comparable to Db2 Version 1 for Linux/UNIX/Windows is performed.

If set to `2`, more optimization is performed than if set to a value of `1`, but significantly less optimization is performed than if set to a value of `3` and higher, particularly for complex queries.

If set to `3`, a moderate amount of optimization is performed.

If set to `5`, a significant amount of optimization is performed. For complex dynamic SQL statements, heuristic rules are used to limit the amount of time spent selecting a query plan. When possible, Select statements obtain data from a materialized query table.

If set to `7`, a significant amount of optimization is performed. For complex dynamic SQL statements, heuristic rules are not used to limit the amount of time spent selecting a query plan. When possible, Select statements obtain data from a materialized query table.

If set to `9`, the maximum amount of optimization is performed, which can significantly expand the number of query plans that are evaluated for selection. This value, along with performance metrics, can be used to determine if a better-performing query plan can be generated for complex or long-running SQL statements.

Data Source Methods

```
public Integer getCurrentQueryOptimization()
```

```
public void setCurrentQueryOptimization(Integer)
```

Default

`-1`

Data Type

Int

See also

Choosing a Db2 optimization class on page 84

Database

Purpose

Specifies the name of the database to which you want to connect. This property is supported only for Db2 for Linux/UNIX/Windows.

Valid Values

`string`

where:

`string`

is the name of a Db2 database.

Data Source Methods

```
public String getDatabase()
public void setDatabase(String)
```

Default

No default value

Data Type

String

DynamicSections

Purpose

Specifies the maximum number of prepared statements that the driver can have open at any one time.

Valid Values

x

where:

x

Is a positive integer that represents a number of prepared statements.

Data Source Methods

```
public Integer getDynamicSections()
public void setDynamicSections(Integer)
```

Default

`200` (prepared statements)

Data Type

Int

EnableCancelTimeout

Purpose

Determines whether a cancel request that is sent by the driver as the result of a query timing out is subject to the same query timeout value as the statement it cancels.

Valid Values

`true` | `false`

If set to `true`, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application calls `Statement.setQueryTimeout(5)` on a statement and that statement is cancelled because its timeout value was exceeded, the driver sends a cancel request that also will time out if its execution exceeds 5 seconds. If the cancel request times out, because the server is down, for example, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid.

If set to `false`, the cancel request does not time out.

Data Source Methods

```
public Boolean getEnableCancelTimeout()
```

```
public void setEnableCancelTimeout(Boolean)
```

Default

`false`

Data Type

Boolean

EncryptionMethod

Purpose

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

Valid Values

`noEncryption` | `DBEncryption` | `requestDBEncryption` | `SSL`

Behavior

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

If set to `DBEncryption`, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the connection fails and the driver throws an exception. The AuthenticationMethod property must be set to a value of `clearText`, `encryptedPassword`, or `encryptedUIDPassword`. This value is not supported for Db2 for i.

If set to `requestDBEncryption`, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the driver attempts to establish an unencrypted connection. The AuthenticationMethod property must be set to a value of `clearText`, `encryptedPassword`, or `encryptedUIDPassword`. This value is not supported for Db2 for i.

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.

Notes

- Connection hangs can occur when the driver is configured for TLS/SSL and the database server does not support TLS/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 TLS/SSL.

Data Source Methods

```
public String getEncryptionMethod()
```

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

Default

`noEncryption`

Data Type

String

See also

Configuring TLS/SSL encryption on page 65

Performance considerations on page 73

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 Methods

```
public String getFailoverGranularity()
```

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

Default

`nonAtomic`

Data Type

String

See also

Failover on page 70

FailoverMode

Purpose

Specifies the type of failover method the driver uses.

Valid Values

`connect` | `extended` | `select`

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 performed by the last Select statement 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

Failover on page 70

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 Methods

```
public Boolean getFailoverPreconnect()
```

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

Default

`false`

Data Type

Boolean

See also

Failover on page 70

Grantee

Purpose

Specifies the name of the schema to which you want to grant EXECUTE privileges for Db2 packages. This property is ignored if the GrantExecute property is set to `false`.

Valid Values

`string`

where:

`string`

Is a valid Db2 schema.

Notes

- Using a value other than `PUBLIC` restricts access to use the driver. For example, if you set this property to `TSMITH`, only the user TSMITH would be allowed access to use the driver against the server.

Data Source Methods

```
public String getGrantee()
```

```
public void setGrantee(String)
```

Default

`PUBLIC`

Data Type

String

See also

Db2 packages on page 76

GrantExecute

Purpose

Determines which Db2 schema is granted EXECUTE privileges for Db2 packages.

Valid Values

`true` | `false`

Behavior

If set to `true`, EXECUTE privileges are granted to the schema that is specified by the Grantee property.

If set to `false`, EXECUTE privileges are granted to the schema that created the Db2 packages.

Data Source Methods

```
public Boolean getGrantExecute()
public void setGrantExecute(Boolean)
```

Default

`true`

Data Type

Boolean

See also

Db2 packages on page 76

GSSPluginName

Purpose

Specifies the name of the GSS plug-in used for authentication when `AuthenticationMethod=pluginSecurity` is enabled.

Valid Values

`string`

where:

`string`

is the name of the GSS plug-in.

Notes

- The plug-in name is case sensitive.
- This value is required when GSS plug-in authentication is enabled (`AuthenticationMethod=pluginSecurity`).

Data Source Methods

```
public String getGSSPluginName()
public void setGSSPluginName(String)
```

Default

No default value

Data Type

String

See also

GSSPluginObject on page 133

GSSPluginObject

Purpose

Specifies the Java object which implements the GSS API authentication plug-in. This property is used only with a dataSource object or a java.util.Properties object supplied through the DriverManager class.

Valid Values

x

where:

x

is a valid object of a plug-in implementation class that extends the abstract `com.ddtek.jdbc.db2.gssplugin.DB2GSSPluginClient` class and provides a valid implementation for the `getTicket()` method.

Notes

- This value is required when GSS plug-in authentication is enabled (`AuthenticationMethod=pluginSecurity`).
- The implementation of this property depends on the type of GSS plug-in enabled on the Db2 server.
- The Java package for this class is updated to *`OEM_pkg_prefix`*`.jdbc.db2.gssplugin.DB2GSSPluginClient` after OEM branding installation.
- The getTicket() method establishes the security context between the driver and the DBMS.
  - The driver calls this method to obtain a GSS token from the client security implementation and sends the token to the DBMS for authentication.
  - The definition of this method is `public abstract byte[] getTicket(String userid, String password, byte[] serverToken) throws SQLException;`

Default

No default value

Data Type

com.ddtek.jdbc.db2.gssplugin.DB2GSSPluginClient implementation object

See also

GSSPluginName on page 132

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.

If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.

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.

Data Source Methods

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

Default

No default value

Data Type

String

See also

- Performance considerations on page 73
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

InitializationString

Purpose

Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Valid Values

command [[; *command*]...]

where:

command

is a SQL command.

Notes

- Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Example

The following connection URL adds USER2 to the CURRENT PATH special register and sets the CURRENT PRECISION special register to DEC31.

```
jdbc:datadirect:db2://server1:50000;InitializationString=
(SET CURRENT PATH=current_path, USER2;SET CURRENT PRECISION='DEC31')
```

Setting the CURRENT PRECISION special register is only valid for Db2 for z/OS.

Data Source Methods

```
public String getInitializationString()
```

```
public void setInitializationString(String)
```

Default

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*

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, the driver pages the result set data to disk. 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*, where *x* is a positive integer that represents the size (in KB) of the memory buffer. The driver caches insensitive result set data in memory and uses this value to set the size of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. 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

`2048`

Data Type

Int

See also

Performance considerations on page 73

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`

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 Methods

```
public Boolean getJavaDoubleToString()
```

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

Default

`false`

Data Type

Boolean

JDBCBehavior

Purpose

Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.

Valid Values

`0 | 1`

Behavior

If set to `0`, the driver describes the data types as JDBC 4.0 data types.

If set to `1`, the driver describes the data types using JDBC 3.0-equivalent data types. This allows your application to continue using JDBC 3.0 types. In addition, the JDBC 4.0 method ResultSet.getHoldability() returns the value of the JDBC 3.0 method Connection.getHoldability().

Data Source Methods

```
public Integer getJDBCBehavior()
```

```
public void setJDBCBehavior(Integer)
```

Default

`1`

Data Type

Int

KeepAlive

Purpose

Specifies whether the driver enables KeepAlive. KeepAlive maintains idle TCP connections by periodically passing packets between the client and server. If either the client or server does not respond to a packet, the connection is considered inactive and is terminated. In addition, KeepAlive prevents valid idle connections from being disconnected by firewalls and proxies by maintaining network activity.

Valid Values

`true | false`

Behavior

If set to `true`, the driver enables KeepAlive.

If set to `false`, the driver does not enable KeepAlive.

Data Source Methods

```
public Boolean getKeepAlive()
```

```
public void setKeepAlive(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

See also

Data encryption on page 64

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

- 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.
- 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

See also

Data encryption on page 64

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.

Valid Values

`string`

where:

`string`

is a valid password.

Notes

- 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.
- The keystore and truststore files can be the same file; therefore, they may have the same password.

Data Source Methods

```
public String getKeyStorePassword()
```

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

Default

No default value

Data Type

String

See also

Data encryption on page 64

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

`false`

Data Type

Boolean

See also

Failover on page 70

LobStreamingProtocol

Purpose

Determines whether streaming or materialization (client caching) is used when the driver fetches LOB and XML data. In most cases, streaming provides better performance; however, materialization can reduce network round trips to the database server when updating LOB data using Clob and Blob objects, which can improve performance.

Valid Values

`streaming` | `materialize`

Behavior

If set to `streaming`, the driver determines the most efficient way to return LOB and XML data based on the following conditions:

- Whether streaming is supported by the database server
- Whether the application requests the data as a stream

For example, if streaming is supported by the database server and an application requests LOB data for a row as a stream, the driver uses streaming to transport the data. If streaming is not supported by the database server or if the application requests the data as a byte array or a String, the driver fully materializes the data on the client.

If set to `materialize`, the driver fully materializes the data on the client instead of using streaming.

Data Source Methods

```
public String getLobStreamingProtocol()
```

```
public void setLobStreamingProtocol(String)
```

Default

`streaming`

Data Type

String

See also

Performance considerations on page 73

LocationName

Purpose

Specifies the name of the Db2 location that you want to access.

For Db2 for z/OS, your system administrator can determine the name of your Db2 location using the following command:

```
DISPLAY DDF
```

For Db2 for i, your system administrator can determine the name of your Db2 location using the following command. The name of the database that is listed as *LOCAL is the value you should use for this property.

```
WRKRDBDIRE
```

This property is not supported for Db2 for Linux/UNIX/Windows.

Valid Values

`string`

where:

string

is the Db2 location.

Data Source Methods

```
public String getLocationName()
```

```
public void setLocationName(String)
```

Default

No default value

Data Type

String

Alias

DatabaseName property. If both the DatabaseName and LocationName connection properties are specified in a connection URL, the last property positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the LocationName property would be used instead of the value of the DatabaseName property.

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;LocationName=acct;
User=test;Password=secret
```

LoginTimeout

Purpose

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

Valid Values

`0` | *x*

where:

x

is a positive integer that represents a number of seconds.

Behavior

If set to `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

0

Data Type

Int

LongDataCacheSize

Purpose

Determines whether the driver caches long data (images, pictures, long text, binary, or XML data) in result sets. To improve performance, you can disable long data caching if your application retrieves columns in the order in which they are defined in the result set.

Valid Values

-1 | 0 | *x*

where:

x

is a positive integer that represents the size of the memory buffer.

Behavior

If set to -1, the driver does not cache long data in result sets. It is cached on the server. Use this value only if your application retrieves columns in the order in which they are defined in the result set.

If set to 0, the driver caches long data in result sets in memory. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

If set to *x*, the driver caches long data in result sets in memory and uses this value to set the size (in KB) of the memory buffer for caching result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

Data Source Methods

```
public Integer getLongDataCacheSize()
```

```
public void setLongDataCacheSize(Integer)
```

Default

2048

Data Type

Int

See also

Performance considerations on page 73

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.

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.

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.

Data Source Methods

```
public Integer getMaxPooledStatements()
```

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

Default

`0`

Data Type

Int

See also

- Performance considerations on page 73
- RegisterStatementPoolMonitorMBean on page 156
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

OptimizationProfile

Purpose

Specifies the Db2 optimization profile that is used during SQL optimization. The optimization profile contains optimization guidelines that are passed to Db2 to influence the generation of query plans. This property sets the value of the Db2 OPTIMIZATION PROFILE special register.

This property only applies to Db2 for Linux/UNIX/Windows.

Valid Values

profile_name

where:

profile_name

is the name of a valid Db2 optimization profile.

Data Source Methods

```
public String getOptimizationProfile()
```

```
public void setOptimizationProfile(String)
```

Default

null

Data Type

String

See also

Performance considerations on page 73

OptimizationProfileToFlush

Purpose

Specifies the Db2 optimization profile to be removed from the optimization profile cache. An optimization profile is processed the first time it is used to optimize a SQL statement, and the output is stored in the optimization profile cache. If the profile is modified after it is processed, its output should be removed from the optimization profile cache.

This property is only applicable to connections to Db2 for Linux/UNIX/Windows.

Valid Values

profile_name | `all`

where:

profile_name

is the name of a valid Db2 optimization profile.

Behavior

If set to *profile_name*, the driver removes the specified profile from the optimization profile cache.

If set to `all`, the driver removes all profiles from the optimization profile cache.

Notes

- The user specifying this property must have administrator privileges to remove optimization profiles from the optimization profile cache. Consult with your system administrator.

Data Source Methods

```
public String getOptimizationProfileToFlush()
public void setOptimizationProfileToFlush(String)
```

Default

`null`

Data Type

String

See also

Performance considerations on page 73

PackageCollection

Purpose

Specifies the name of the collection or library (group of packages) to which Db2 packages are bound.

Valid Values

string

where:

string

is the name of the collection or library (group of packages) to which Db2 packages are bound.

Notes

- This property is ignored for Db2 for Linux/UNIX/Windows.
- This property replaces the CollectionId property, which has been deprecated; however, the CollectionId property is still recognized for backward compatibility. If both the PackageCollection and CollectionId properties are specified, the CollectionId property is ignored.

Data Source Methods

```
public String getPackageCollection()
```

```
public void setPackageCollection(String)
```

Default

`NULLID`

Data Type

String

See also

Db2 packages on page 76

PackageOwner

Purpose

Specifies the owner to be used for any Db2 packages that are created.

Valid Values

`string`

where

`string`

is the owner of the Db2 packages.

Data Source Methods

```
public String getPackageOwner()
```

```
public void setPackageOwner(String)
```

Default

`NULL`

Data Type

String

See also

Db2 packages on page 76

Password

Purpose

Specifies a password that is used to connect to your Db2 database. A password is required if security is enabled on your database. Contact your system administrator to obtain your password.

Valid Values

string

where

string

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

Data Source Methods

```
public String getPassword()
```

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

Default

No default value

Data Type

String

PortNumber

Purpose

Specifies the TCP port of the primary database server that is listening for connections to the Db2 database.

This property is supported only for data source connections.

Valid Values

port

where:

port

is the port number.

Data Source Methods

```
public Integer getPortNumber()
```

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

Default

```
50000
```

Data Type

Int

ProgramID

Purpose

Specifies the driver and version information on the client to be stored in the database. This property sets the CLIENT_PRDID value in the database. For Db2 for Linux/UNIX/Windows, this value is located in the SYSIBMADM.APPLICATIONS table.

Valid Values

`DDJ` *`VVRRM`*

where:

`DDJ`

identifies the driver as a DataDirect Connect Series for JDBC driver.

`VV`

identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).

`RR`

identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).

`M`

identifies a 1-character modification level (0-9 or A-Z).

Example

```
DDJ04200
```

Data Source Methods

```
public String getProgramID()
```

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

Default

No default value

Data Type

String

See also

Client information on page 72

ProxyHost

Description

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. See "IP addresses" for details about using these formats.

Data Source Methods

```
public String getProxyHost()
```

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

Default

No default value

Data Type

String

See also

IP addresses on page 69

Proxy server on page 69

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

No default value

Data Type

String

See also

Proxy server on page 69

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

`0`

Data Type

Int

See also

Proxy server on page 69

ProxyUser

Purpose

Specifies the 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

No default value

Data Type

String

See also

Proxy server on page 69

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 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

`0`

Data Type

Int

RandomGenerator

Purpose

Specifies the type of random number generator the database uses for secure seeding. Db2 uses a random number generator for secure seeding of data encrypted with the AES algorithm. AES encryption can be implemented with the AuthenticationMethod connection property.

Valid Values

`random` | `secureRandom`

Behavior

If set to `random`, a stream of pseudorandom numbers is generated for secure seeding.

If set to `secureRandom`, a cryptographically strong number generation algorithm is used for secure seeding. By default, the SHA1PRNG algorithm is used. You can also use the SecureRandomAlgorithm connection property to designate the number generation algorithm you want to use.

Notes

- When establishing a connection with a connection string, RandomGenerator should precede the User and Password connection properties in the connection URL. When using a data source connection, RandomGenerator should be set before making calls to setUser(), setPassword(), or setNewPassword().
- `SecureRandom` offers more secure seeding of random numbers, and, in turn, requires additional processing. Therefore, response times for applications may be slower when compared to response times with RandomGenerator set to `random`. By default, the driver is set to `secureRandom`.
- For more information on SecureRandom seeding, refer to the API specification for your edition of Java.

Data Source Methods

```
public String getRandomGenerator()
```

```
public void setRandomGenerator(String)
```

Default

```
secureRandom
```

Data Type

String

See also

- SecureRandomAlgorithm on page 158
- AuthenticationMethod on page 111
- Random number generator secure seeding on page 55
- Performance considerations on page 73

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.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMBean()
```

```
public void setRegisterStatementPoolMonitorMBean(Boolean)
```

Default

```
false
```

Data Type

Boolean

See also

- MaxPooledStatements on page 146
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

ReplacePackage

Purpose

Determines whether the current bind process will replace the existing Db2 packages used by the driver.

For Db2 for Linux/UNIX/Windows, this property must be used in conjunction with the CreateDefaultPackage property.

Valid Values

`true` | `false`

Behavior

If set to `true`, the current bind process will replace the existing Db2 packages that are used by the driver.

If set to `false`, the current bind process will not replace the existing Db2 packages.

Data Source Methods

```
public Boolean getReplacePackage()
```

```
public void setReplacePackage(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Db2 packages on page 76

ResultSetMetaDataOptions

Purpose

Determines whether the driver returns table name information in the ResultSet metadata for Select statements.

Valid Values

`0 | 1`

Behavior

If set to `0` and the ResultSetMetaData.getTableName() method is called, the driver does not perform additional processing to determine the correct table name for each column in the result set. The getTableName() method may return an empty string for each column in the result set.

If set to `1` and the ResultSetMetaData.getTableName() method is called, the driver performs additional processing to determine the correct table name for each column in the result set. The driver returns schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information.

Data Source Methods

```
public Integer getResultSetMetaDataOptions()
```

```
public void setResultSetMetaDataOptions(Integer)
```

Default

`0`

Data Type

Int

See also

Performance considerations on page 73

SecureRandomAlgorithm

Purpose

Specifies the SecureRandom number generation algorithm used for secure seeding when the RandomGenerator connection property is set to `secureRandom`.

Valid Values

`algorithm_name`

where:

`algorithm_name`

is the name of a SecureRandom number generation algorithm supported by the JDK packaged with your database management system.

Example

The connection property is a key-value pair with the name of the SecureRandom number generation algorithm written as a string, for example, `SecureRandomAlgorithm=SHA1PRNG`.

Notes

- SecureRandomAlgorithm can only be used when RandomGenerator is set to `secureRandom`. If `RandomGenerator=secureRandom` and no value is specified for SecureRandomAlgorithm, the SHA1PRNG algorithm is used for secure seeding.
- When establishing a connection with a connection string, RandomGenerator and SecureRandomAlgorithm should precede the User and Password connection properties in the connection URL. When using a data source connection, RandomGenerator and SecureRandomAlgorithm should be set before making calls to setUser(), setPassword(), or setNewPassword().
- Refer to your database management system documentation to see which SecureRandom number generation algorithms are included in the JDK packaged with your system. Additional information is also available on the Java Cryptography Architecture Standard Algorithm Name Documentation for JDK 8 Web page.

Data Source Methods

```
public String getSecureRandomAlgorithm()
```

```
public void setSecureRandomAlgorithm(String)
```

Default

No default value

Data Type

String

See also

- RandomGenerator on page 155
- Random number generator secure seeding on page 55

SendStreamAsBlob

Purpose

Determines whether binary stream data that is less than 32 KB is sent to the database as Db2 Long Varchar for Bit Data or Blob data. Binary streams that are larger than 32 KB can only be inserted into a Blob column. The driver always sends binary stream data larger than 32 KB to the database as Blob data.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver sends binary stream data that is less than 32 KB to the database as Db2 Blob data. If the target column is a Long Varchar for Bit Data column and not a Blob column, the Insert or Update statement fails. The driver automatically retries the Insert or Update statement, sending the data as Long Varchar for Bit Data, if the pointer in the stream can be reset to the beginning of the stream. If you know that you are sending the binary stream data to a Blob column, setting this value improves performance.

If set to `false`, the driver sends binary stream data that is less than 32 KB to the database as Long Varchar for Bit Data data. If the target column is a Blob column and not a Long Varchar for Bit Data column, the Insert or Update statement fails. The driver retries the Insert or Update statement, sending the data as Blob data, if the pointer in the stream can be reset to the beginning of the stream.

Data Source Methods

```
public Boolean getSendStreamAsBlob()
```

```
public void setSendStreamAsBlob(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Performance considerations on page 73

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.

This property is supported only for data source connections.

Valid Values

`string`

where:

`string`

is a valid IP address or server name.

Example

`122.23.15.12` or `DB2Server`

Data Source Methods

```
public String getServerName()
```

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

Default

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).
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.

Attribute	Description
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.
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`.
- If a log file name does not include the `.log` extension, the driver automatically appends it. For example, a file named `spy.jsp` is renamed to `spy.jsp.log` by the driver.
- Refer to "Tracking JDBC Calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference* for a list of supported attributes.

Data Source Methods

```
public String getSpyAttributes()
```

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

Default

No default value

Data Type

String

StripNewlines

Purpose

Specifies whether newline characters in a SQL statement are sent to the Db2 server. If you know that SQL statements in your application do not contain newline characters, the driver can skip the removal of newline characters, improving performance by eliminating the parsing required by the Db2 server to remove them.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver removes all newline characters from SQL statements.

If set to `false`, the driver does not remove any newline characters from SQL statements.

Data Source Methods

```
public Boolean getStripNewlines()
```

```
public void setStripNewlines(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Performance considerations on page 73

TrustStore

Purpose

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

Valid Values

`string`

where:

`string`

is the directory of the truststore file.

Notes

- 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`.

Data Source Methods

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

Default

No default value

Data Type

String

See also

Data encryption on page 64

TrustStorePassword

Purpose

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

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

Notes

- 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`.

Data Type

String

See also

Data encryption on page 64

UseCurrentSchema

Purpose

Specifies whether results are restricted to the tables and views in the current schema if a DatabaseMetaData.getTables() or DatabaseMetaData.getColumns() method is called without specifying a schema or if the schema is specified as the wildcard character %. Restricting results to the tables and views in the current schema improves the performance of calls for getTables() methods that do not specify a schema.

Valid Values

`true` | `false`

Behavior

If set to `true`, results that are returned from the getTables() and getColumns() methods are restricted to tables and views in the current schema.

If set to `false`, results of the getTables() and getColumns() methods are not restricted.

Data Source Methods

```
public Boolean getUseCurrentSchema()
```

```
public void setUseCurrentSchema(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Performance considerations on page 73

User

Purpose

Specifies the user name that is used to connect to the Db2 database.

Valid Values

`string`

where:

`string`

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

Data Source Methods

```
public String getUser()
```

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

Default

No default value

Data Type

String

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

WithHoldCursors

Purpose

Determines whether the cursor stays open on commit. After a commit, Db2 can leave all cursors open (Preserve cursors) or close all open cursors (Delete cursors). Rolling back a transaction closes all cursors regardless of how this property is specified.

Valid Values

`true` | `false`

Behavior

If set to `true`, the cursor behavior is Preserve.

If set to `false`, the cursor behavior is Delete.

Data Source Methods

```
public Boolean getWithHoldCursors()
public void setWithHoldCursors(Boolean)
```

Default

`true`

Data Type

Boolean

XMLDescribeType

Purpose

Determines whether the driver maps XML data to the CLOB or BLOB data type.

Valid Values

`clob` | `blob`

Behavior

If set to `clob`, the driver maps XML data to the CLOB data type.

If set to `blob`, the driver maps XML data to the BLOB data type.

Data Source Methods

```
public String getXMLDescribeType()
public void setXMLDescribeType(String)
```

Default

No default value

Data Type

String

See also

Returning and inserting/updating XML data on page 79