Progress DataDirect

Progress DataDirect for JDBC for Microsoft SQL Server

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Microsoft SQL Server Driver

The Progress® DataDirect® for JDBC ™ for Microsoft SQL Server™ driver supports the JDBC API for SQL read-write access to:

- Microsoft Azure, including Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System
- Microsoft Fabric
- Microsoft SQL Server

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-microsoft-sql-server.

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 the 6.0.0 release

- **Driver Enhancements**
  - The driver now provides read-write access to the Synapse Data Warehouse endpoints of Microsoft Fabric. For connection URL examples and limitations, see Connection URL examples on page 14 and Support for Microsoft Fabric on page 70.
  - 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 67 for details.
  - The driver has been enhanced to support managed identity authentication for Microsoft Entra ID (Azure Active Directory) resources. You can configure this authentication using the updated AuthenticationMethod and User connection properties. See Microsoft Entra ID authentication on page 51 for details.
  - The driver has been enhanced to support the TLSv1.3 cryptographic protocol using SQL Server 2022. To enable TLSv1.3 encryption, set the EncryptionMethod property to the new value `Strict`. See EncryptionMethod on page 144 for details.
  - The driver has been enhanced to support inserts into IDENTITY columns for data replication. You can enable inserts into IDENTITY columns defined as NOT FOR REPLICATION using the new EnableReplicationUser connection property. See Inserts into IDENTITY columns for data replication on page 102 and EnableReplicationUser for details.
  - The driver has been enhanced to support Microsoft Entra ID authentication using service principal users. You can configure this feature using the refreshed AuthenticationMethod property, and the new ActiveDirectoryPrincipalID and ActiveDirectoryPrincipalSecret properties. See Microsoft Entra ID authentication on page 51 for details.
  - The driver has been enhanced to support authentication using an access token. See Access token authentication on page 49 for details.
  - The driver has been enhanced to support Windows Defender Credential Guard when using Kerberos Authentication. See Kerberos authentication on page 54 for details.

- The driver has been enhanced to support encrypted parameters in stored procedures when using the Always Encrypted feature.
- The driver has been enhanced to support the Always Encrypted feature. Beginning with SQL Server 2016, Azure SQL and SQL Server databases support Always Encrypted, which allows sensitive data to be stored on the server in an encrypted state such that the data can only be decrypted by an authorized application. The following are highlights of this enhancement:
  - The driver detects all supported native data types stored in encrypted columns and transparently encrypts values bound to SQL parameters or decrypts values returned in results.
  - The driver supports configurable caching of column encryption keys for improved performance.
  - The driver supports using Java KeyStore and Azure Key Vault as keystore providers.

  You can enable support for Always Encrypted using the following new options: ColumnEncryption, AEKeyCacheTTL, AEKeystoreClientSecret, AEKeystoreLocation, AEKeystorePrincipalId, and AEKeystoreSecret. See Always Encrypted on page 64 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 175 for details.
  - HostProcess is no longer supported as an alias for the ProgramID connection property.
  - ProgramName is no longer supported as an alias for the ApplicationName connection property.
  - The driver has been updated to require a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions. See Requirements for more details on JVM.

    Java SE 7 has reached the end of its product life cycle and will no longer receive generally available security updates. As a result, the driver will no longer support JVMs that are version Java SE 7 or earlier. Support for distributed versions of Java SE 7 and earlier will also end.
  - Microsoft has rebranded Azure Active Directory as Microsoft Entra ID. As a result, the driver documentation has been updated to reflect this name change.

Changes for the 6.0.0 release

- **Driver Enhancements**
  - The driver has been enhanced to transparently connect to Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System data sources. See Azure Synapse Analytics and Analytics Platform System on page 68 for more information about supported features and functionality.
  - The driver has been enhanced to support Always On Availability Groups. Introduced in SQL Server 2012, Always On Availability Groups is a replica-database environment that provides a high-level of data availability, protection, and recovery. See Always On Availability Groups on page 77 for details on using the driver with this feature.
  - The driver is enhanced to support Microsoft Entra ID authentication using user names and passwords. Entra ID authentication is an alternative to SQL Server Authentication that allows administrators to centrally manage user permissions to Azure SQL Database data stores. See Microsoft Entra ID authentication on page 51 for details.
  - The driver has been enhanced to support Kerberos constrained delegation. Constrained delegation is a Kerberos mechanism that allows a client application to delegate authentication to a second service. See Kerberos authentication on page 54 and Constrained delegation on page 58 for details.
- **Changed Behavior**

- For Kerberos authentication environments, the following changes have been implemented.
  - The driver no longer sets the java.security.auth.login.config system property to force the use of the installed `JDBCDriverLogin.conf` file as the JAAS login configuration file. By default, the driver now uses the default JAAS login configuration file for Java, unless you specify a different location and file using the java.security.auth.login.config system property.
  - 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.

  See Kerberos authentication on page 54 for details.

Requirements

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

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

Note: To use the driver on a Java Platform with standard Security Manager enabled, certain permissions must be set in the security policy file of the Java Platform.

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\sqlserver.jar
     ```

- **UNIX/LINUX Example**

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/sqlserver.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 either NTLM or user ID and password authentication.

  NTLM

```
jdbc:datadirect:sqlserver://hostname:port;
DatabaseName=database_name;AuthenticationMethod=ntlm;
LoadLibraryPath=lib_path;User=user_name;
Password=password;
```

  UserID/Password

```
jdbc:datadirect:sqlserver://hostname:port;
AuthenticationMethod=userIdPassword;User=user_name;
Password=password;
```

  Note: See Authentication on page 48 to know more about user ID/password authentication and other authentication methods you can use to connect to the server.

- **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

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

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:

- Connection URL Examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
- Connection Property Descriptions provides a complete list of supported properties by functionality.
- Performance Considerations describes connection properties that affect performance, along with recommended settings.

5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:

- Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
- DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.

This completes the deployment of the driver.

See also

Connection property descriptions on page 103

Data source and driver classes

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

Driver class:

com.ddtek.jdbc.sqlserver.SQLServerDriver

DataSource class:

com.ddtek.jdbcx.sqlserver.SQLServerDataSource

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}}`.

```
jdbc:datadirect:sqlserver://servername:port;
[property=value[;...]];
```

- Microsoft Fabric
- Microsoft SQL Server

Microsoft Fabric

Note: For Microsoft Fabric, only the following authentication methods are supported: Microsoft Entra ID user ID/password authentication and Microsoft Entra ID service principal authentication.

- Microsoft Entra ID user ID/password authentication
- Microsoft Entra ID service principal authentication
- Proxy server
- **Microsoft Entra ID user ID/password authentication**:

This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) user ID and password authentication.

```
jdbc:datadirect:sqlserver://myserver.database.windows:1433;
     AuthenticationMethod=activeDirectoryPassword;
     User=jsmith@mydomain.com;Password=secret;
```

For more information on these properties and values, see Microsoft Entra ID authentication on page 51.

- **Microsoft Entra ID service principal authentication**:

  This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) service principal authentication.

  ```
  jdbc:datadirect:sqlserver://myserver.database.windows:1433;
       AuthenticationMethod=activeDirectoryServicePrincipal;
       ActiveDirectoryPrincipalID=789f8b4c-7a4a-445d-6oe9;
       ActiveDirectoryPrincipalSecret=ABcdEFg/hiJkLmNOPqR01stUvW;
  ```

  For more information on these properties and values, see Microsoft Entra ID authentication on page 51.

- **Proxy server**:

  This string includes the properties used to connect through a proxy server.

  ```
  jdbc:datadirect:fabric://myserver:1433;AuthenticationMethod=activeDirectoryPassword;
       ProxyHost=pserver;ProxyPassword=secret;ProxyPort=808;ProxyUser=johnSmith;
       User=jsmith@example.com;Password=secret;
  ```

  For more information on these properties and values, see Connection property descriptions on page 103.

Microsoft SQL Server

- User ID/Password authentication
- Microsoft Entra ID user ID/password authentication
- Microsoft Entra ID service principal authentication
- Microsoft Entra ID managed identity authentication
- Kerberos authentication
- NTLM authentication
- Access token authentication
- Proxy server
- **User ID/password authentication**:

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

  ```
  jdbc:datadirect:sqlserver://myserver:1433;
       AuthenticationMethod=userIdPassword;User=JSmith;
       Password=secret;
  ```

  For more information on these properties and values, see User ID/password authentication on page 49.

- **Microsoft Entra ID user ID/password authentication**:

This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) user ID and password authentication.

```
jdbc:datadirect:sqlserver://myserver.database.windows:1433;
     AuthenticationMethod=activeDirectoryPassword;
     User=jsmith@mydomain.com;Password=secret;
```

For more information on these properties and values, see Microsoft Entra ID authentication on page 51.

- **Microsoft Entra ID service principal authentication**:

  This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) service principal authentication.

```
jdbc:datadirect:sqlserver://myserver.database.windows:1433;
     AuthenticationMethod=activeDirectoryServicePrincipal;
     ActiveDirectoryPrincipalID=789f8b4c-7a4a-445d-6oe9;
     ActiveDirectoryPrincipalSecret=ABcdEFg/hiJkLmNOPqR01stUvW;
```

  For more information on these properties and values, see Microsoft Entra ID authentication on page 51.

- **Microsoft Entra ID managed identity authentication**:

  This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) managed identity authentication.

```
jdbc:datadirect:sqlserver://myserver.database.windows:1433;
     AuthenticationMethod=ActiveDirectoryManagedIdentity;
     User=f4f42d67-6857-4d58-a510-83a3dc082469;
```

  For more information on these properties and values, see Microsoft Entra ID authentication on page 51.

- **Kerberos authentication**:

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

```
jdbc:datadirect:sqlserver://myserver:1433;
     DatabaseName=sqlserverDB;AuthenticationMethod=kerberos;
     ServicePrincipalName=MSSQLSvc/myserver.example.com:1433@EXAMPLE.COM;
```

  For more information on these properties and values, see Kerberos authentication on page 54.

- **NTLM authentication**:

  This string includes the properties used to connect with NTLM authentication with credentials.

```
jdbc:datadirect:sqlserver://myserver:1433;AuthenticationMethod=ntlm;
     Domain=sso3;User=jsmith;Password=secret;
```

  This string includes the properties used to connect with NTLM authentication without credentials (Windows only).

```
jdbc:datadirect:sqlserver://myserver:1433;AuthenticationMethod=ntlm;
     DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;User=jsmith;
     Password=secret;
```

  For more information on these properties and values, see NTLM authentication on page 60.

- **Access token authentication**:

  This string includes the properties used to connect with access token authentication.

```
jdbc:datadirect:sqlserver://myserver:1433;AuthenticationMethod=auto;
     AccessToken=abc12cd34efg5678h9ij87klm6543no32pqr10;
```

For more information on these properties and values, see Access token authentication on page 49.

- **Proxy server**:

  This string includes the properties used to connect through a proxy server.

```
jdbc:datadirect:sqlserver://myserver:1433;AuthenticationMethod=userIdPassword;
     ProxyHost=pserver;ProxyPassword=secret;ProxyPort=808;ProxyUser=johnSmith;
     User=jsmith@example.com;Password=secret;
```

  For more information on these properties and values, see Connection property descriptions on page 103.

See also

Connection property descriptions on page 103
Authentication on page 48

Data types

The following table lists supported data types supported and how they are mapped to JDBC data types.

Table 1: Microsoft SQL Server Data Types

Microsoft SQL Server Data Type	JDBC Data Type
bigint	BIGINT
bigint identity	BIGINT
binary	BINARY
bit	BIT
char	CHAR
date	DATE
datetime	TIMESTAMP
datetime2	TIMESTAMP
datetimeoffset[1]	VARCHAR or TIMESTAMP
decimal	DECIMAL
decimal() identity[2]	DECIMAL
float	FLOAT
image[2]	LONGVARBINARY

[1] 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.
[2] Not supported for Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System.

Microsoft SQL Server Data Type	JDBC Data Type
int	INTEGER
int identity	INTEGER
money	DECIMAL
nchar	NCHAR
ntext[2]	LONGNVARCHAR
numeric	NUMERIC
numeric() identity[2]	NUMERIC
nvarchar	NVARCHAR
nvarchar(max)	LONGNVARCHAR
real	REAL
smalldatetime	TIMESTAMP
smallint	SMALLINT
smallint identity[2]	SMALLINT
smallmoney	DECIMAL
sql_variant[2]	VARCHAR
sysname	VARCHAR
text[2]	LONGVARCHAR
time[3]	TIME or TIMESTAMP
timestamp	BINARY
tinyint	TINYINT
tinyint identity[2]	TINYINT
uniqueidentifier	CHAR
varbinary	VARBINARY
varbinary(max)	LONGVARBINARY

[3] When FetchTWFSasTime=true, this data type is mapped to the JDBC TIME data type. When FetchTWFSasTime=false (the default), this data type is mapped to the JDBC TIMESTAMP data type.

Microsoft SQL Server Data Type	JDBC Data Type
varchar	VARCHAR
varchar(max)	LONGVARCHAR
xml[2,4]	SQLXML

getTypeInfo

The following table provides `getTypeInfo()` results for supported data types.

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 = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = bigint identity AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bigint identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

[4] The XMLDescribeType property overrides the mappings for XML data.

TYPE_NAME = binary AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = binary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8000 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = bit AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -7 (BIT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bit MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char AUTO_INCREMENT = NULL CASE_SENSITIVE = false 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 = 8000 SEARCHABLE = 3 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 = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = date MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = datetime MAXIMUM_SCALE = 3	MINIMUM_SCALE = 3 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 23 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime2 AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = datetime2 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 27 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = datetimeoffset AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) or 93 (TIMESTAMP)[5] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = datetimeoffset MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 34 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
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 = 38	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = decimal() identity[6] AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = *precision* DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal() identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

[5] When FetchTSWTZasTimestamp=false, the data type that is returned by DATA_TYPE is VARCHAR; when FetchTSWTZasTimestamp=true, the data type that is returned is TIMESTAMP.

[6] Not supported for Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System.

TYPE_NAME = float AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 6 (FLOAT) 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 = 2 PRECISION = 53 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = image[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = image MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = int 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 = int 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

TYPE_NAME = int identity AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = int identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = money AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = true LITERAL_PREFIX = $ LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = money MAXIMUM_SCALE = 4	MINIMUM_SCALE = 4 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nchar AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -15 (NCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = nchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 4000 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = ntext[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -16 (LONGNVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = ntext MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 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 = 38[7]	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = numeric() identity[6] AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = *precision* DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = numeric() identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

[7] This value can be configured with a server option in SQL Server and Azure.

TYPE_NAME = nvarchar AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *max length* DATA_TYPE = -9 (NVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = nvarchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 4000 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = nvarchar(max) AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -16 (LONGNVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = nvarchar(max) MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
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 = real 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

TYPE_NAME = smalldatetime AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = smalldatetime MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
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 = smallint identity[6] AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallint identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

TYPE_NAME = smallmoney AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = true LITERAL_PREFIX = $ LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallmoney MAXIMUM_SCALE = 4	MINIMUM_SCALE = 4 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = sql_variant[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = sql_variant MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 8000 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = sysname AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = sysname MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = NULL PRECISION = 128 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = text[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = text 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 = time AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = time MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = timestamp AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = timestamp MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = tinyint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = tinyint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = tinyint identity[6] AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = tinyint identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = uniqueidentifier AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 1(CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = uniqueidentifier MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 36 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = varbinary AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *max length* DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = varbinary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8000 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varbinary(max) AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = varbinary(max) MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar AUTO_INCREMENT = NULL CASE_SENSITIVE = false 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 = 8000 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = varchar(max) AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar(max) 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 = xml[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 2009 (SQLXML) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = xml MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Driver specifications

This section describes the general functionality supported by the driver.

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

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

- **Error handling**: The driver reports errors to the application by throwing SQLExceptions. Each SQLException contains the following information:

  - Description of the probable cause of the error, prefixed by the component that generated the error

- Native error code (if applicable)
- String containing the XOPEN SQLstate

- **Isolation and lock levels**: The SQL Server driver supports the following isolation levels for Microsoft SQL Server.

  Note: For Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System, Read Uncommitted is the only supported isolation level.

  - Read Committed with Locks or Read Committed
  - Read Committed with Snapshots
  - Read Uncommitted
  - Repeatable Read
  - Serializable
  - Snapshot

  For Microsoft SQL Server, the default is Read Committed with Locks or Read Committed.

- **Timeouts**: The driver allows you to impose limits on the duration of active sessions through the use of the EnableCancelTimeout and QueryTimeout connection properties. With the LoginTimeout connection property, you can specify how long the driver waits for a connection to be established before timing out the connection request.

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 sqlserver.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:1433
   ```

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

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

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

   ```
   SELECT * FROM EXAMPLE_TABLES;
   ```

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

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

   The results of the command are displayed in the terminal.

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

   ```
   DISCONNECT
   ```

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

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:

   ```
   sqlserver.jar
   ```

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

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

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

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

     ```
     jdbc:datadirect:sqlserver://MyServer:1433;User=JSmith;Password=secret;
     ```

     Note: See User ID/password authentication on page 49 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **User**: 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\sqlserver.jar
   ```

   Linux

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

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

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

  ```
  SQLServer
  ```

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

  ```
  jdbc:datadirect:sqlserver://MyServer:1433
  ```

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

  ```
  com.ddtek.jdbc.sqlserver.SQLServerDriver
  ```

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

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

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

     ```
     jdbc:datadirect:sqlserver://MyServer:1433;User=JSmith;Password=secret;
     ```

   Note: See Authentication on page 48 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 `TESTNEWTIME` table:

   ```
   SELECT * FROM EMPLOYEE
   ```

To select the URLs for a specified issue:

```
SELECT * FROM EMPLOYEE WHERE ID = <EMPID>
```

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

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

You have successfully accessed your data with DbVisualizer.

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/sqlserver.jar
```

Windows Example

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

UNIX Example

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

Connecting using the JDBC Driver Manager

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

UserID/Password authentication

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret;DatabaseName=MyDB");
```

Note: See User ID/password authentication on page 49 for details.

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with UserID/Password authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:sqlserver://hostname:port;User=user_name;
Password=password;[property=value[;...]]
```

where:

hostname

is the IP address or host name of the server to which you are connecting.

port

is the number of the TCP/IP port.

user_name

Specifies the user ID used to authenticate to SQL Server with UserID/Password authentication method.

password

Specifies the password used to authenticate to SQL Server with UserID/Password authentication method.

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

property=value

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

The following example connection string includes the properties required for connecting with the UserID/Password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:sqlserver://MyServer:1433;
 User=test;Password=secret;DatabaseName=MyDB");
```

See also

Connection property descriptions on page 103
Connection URL examples on page 14

Connecting using data sources

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

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

How data sources are implemented

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

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

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

See also

Data source and driver classes on page 14

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.sqlserver.SQLServerDataSource;
```

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

Note:

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

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQL Server Datasource");
mds.setServerName("MyServer");
mds.setPortNumber(1433);
mds.setUser("test");
mds.setPassword("secret");
```

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

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

Calling a data source in an application

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

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

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

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

Testing a data source connection

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

Take the following steps to establish a connection.

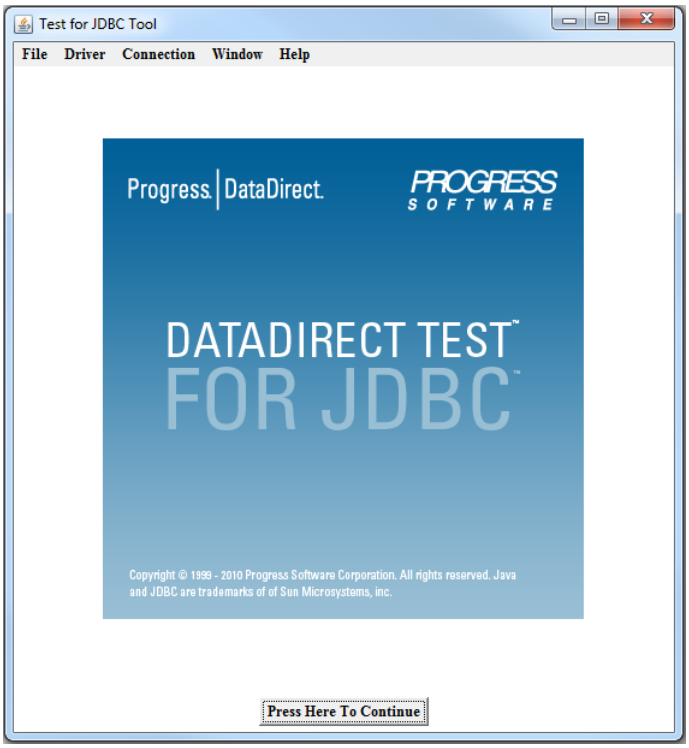
1. Navigate to the installation directory. The default location is:

- Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
- UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

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

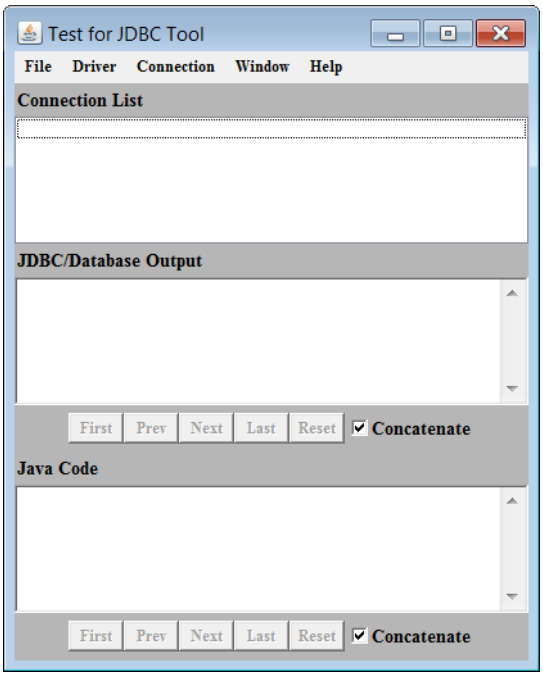
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



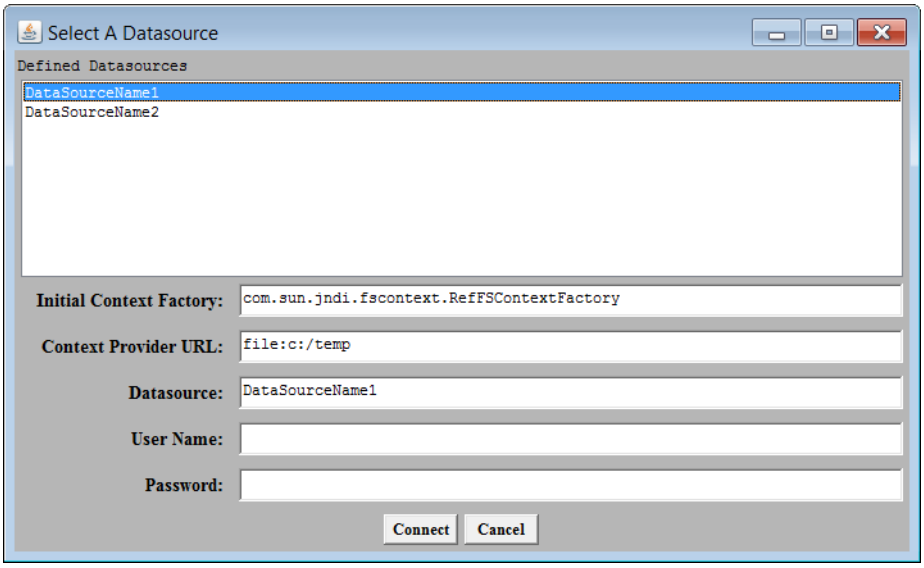
3. Click **Press Here to Continue**.

   The main dialog appears:



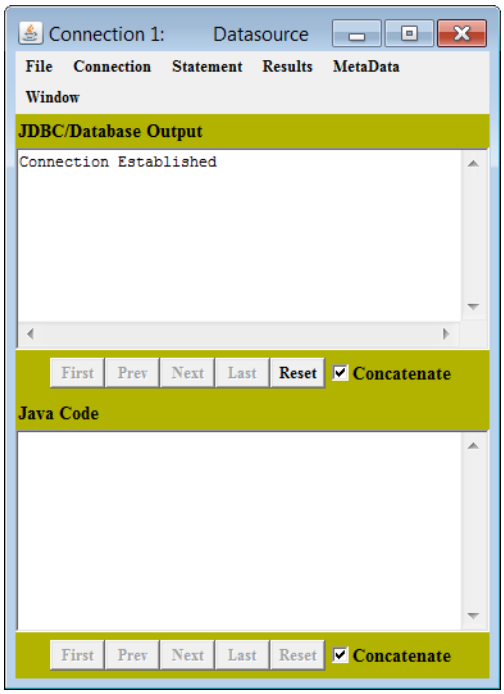
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



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

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



Authentication

Depending on the authentication mechanism used in your environment, additional steps may be required to configure authentication.

The driver supports the following authentication methods:

- *UserID/Password* authenticates using the specified user IDs and passwords.
- *Kerberos* authenticates based on the value specified for the LoginConfigName property to establish a Kerberos connection.
- *NTLM* authenticates using NTLMv1 or NTLMv2 depending on the size of the NTLM password.
- *Access token* authenticates the user with an access token obtained from the Microsoft Entra ID (Azure Active Directory).
- *Microsoft Entra ID authentication* (formerly Azure Active Directory Authentication) authenticates using Microsoft Entra ID (Entra ID) authentication when establishing a connection to Azure. The driver supports the following methods of Entra ID authentication:
  - User and password authentication
  - Service principal user authentication
  - Managed identity authentication

By default, the driver is configured to use the user ID/password authentication, access token authentication, or Kerberos authentication based on the connection properties provided (`AuthenticationMethod=auto`).

See also

User ID/password authentication on page 49
Access token authentication on page 49
Microsoft Entra ID authentication on page 51
Kerberos authentication on page 54
NTLM authentication on page 60

User ID/password authentication

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

1. Set the AuthenticationMethod property to `userIdPassword` or `auto`.
2. Set the User property to provide the user ID.
3. Set the Password property to provide the password.
4. Specify values for minimum required properties for establishing a connection.
   a) Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server.
   b) Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

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

```
Connection conn = DriverManager.getConnection
          ("jdbc:datadirect:sqlserver://server1:1433;
          AuthenticationMethod=userIdPassword;User=test;
          Password=secret);
```

See also

AuthenticationMethod on page 125
User on page 182
Password on page 163

Access token authentication

The driver supports access token authentication with a token obtained from the Microsoft Entra ID (Azure Active Directory). The driver sets the AuthenticationMethod property to `auto`, if an access token is specified. Configuring access token authentication is recommended using a `Properties` or `DataSource` object only. The driver uses the SSL value for EncryptionMethod property when authenticating via access token.

Note: If an access token is specified, it will take precedence over other authentication methods.

An access token can be generated using MSAL4J or external libraries. Alternatively, a token can be generated using the driver class `SQLServerMSAL4JUtil` as shown in the following example.

```
import com.ddtek.jdbc.sqlserver.SQLServerMSAL4JUtils;

String spn="https://database.windows.net/";
String stsurl="https://login.microsoftonline.com/stsurl";// Replace with your STS URL.
String clientId="ab123c45-def6-7gm2345no67891";// Replace with your client ID.
String clientSecret="12a3=bCD/Gh4Ijk+Lm7qR8s=//TuV+Wd";// Replace with your client secret.

String accessToken="";
SQLServerMSAL4JUtils adal4jUtils=new SQLServerMSAL4JUtils();
try {
  accessToken=adal4jUtils.generateFedAuthAccessToken(
  spn,
  stsurl,
  clientId,
  clientSecret.toCharArray());}
catch (Exception exception)
 {
  System.err.println("exception in access token generation:"+exception.getMessage());
 }
```

The following example shows the configuration of access token authentication using a datasource object.

```
SQLServerDataSource ds=new SQLServerDataSource();

ds.setServerName("myserver.windows.net");
ds.setDatabaseName("mydatabase");
ds.setAccessToken(accessToken);

try (Connection connection=ds.getConnection();
Statement stmt=connection.createStatement();
ResultSet rs=stmt.executeQuery("SELECT USER_NAME()"))
{
 if (rs.next())
 {
  System.out.println("You have successfully logged on as: " + rs.getString(1));
 }
}
```

If the connection is successful, the following message is displayed:

```
Access Token: access_token
You have successfully logged on as client_id.
```

See also

AuthenticationMethod on page 125
AccessToken on page 114
Connection property descriptions on page 103

Microsoft Entra ID authentication

The driver supports Microsoft Entra ID (Entra ID) authentication (formerly known as Azure Active Directory authentication). Entra ID authentication is an alternative to SQL Server Authentication that allows administrators to centrally manage user permissions to Azure SQL Database data stores. The driver supports the following methods of Entra ID authentication:

- User and password authentication: The driver authenticates using the Entra ID user and password.
- Service principal user authentication: The driver retrieves an access token by authenticating using the principal id of the logical server and the client secret of your Entra ID application.
- Managed identity authentication: The driver authenticates using a system-assigned or user-assigned managed identity. Managed identities are a type of service principal that can be used only with Azure resources for which they are granted permissions. Using managed identities to authenticate provides an alternative to using credentials, and they can be used anywhere Entra ID authentication is supported.

Note: When using Entra ID authentication, the driver requires root CA certificates to establish an SSL connection to a database. The driver determines the location of the truststore containing the required certificates by using the default JRE `cacerts` file unless a different file has been specified by the `javax.net.ssl.trustStore` Java system property. The truststore location cannot be specified using the driver's Truststore property.

User and password authentication

To use user and password authentication with Entra ID:

- Set the AuthenticationMethod property to specify a value of `ActiveDirectoryPassword`.
- Set the User property to specify your Entra ID username using the `userid@domain.com` format.
- Set the Password property to specify your Entra ID password.
- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, `myserver.database.windows.net`.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

For example, the following is a connection string with only the required options for making a connection using Entra ID authentication.

Note: If the HostNameInCertificate is not specified, the driver automatically uses the value of the ServerName from the URL as the value for validating the certificate.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:sqlserver://your_server.database.windows:1433;
AuthenticationMethod=ActiveDirectoryPassword;
User=test@mydomain.com;Password=secret");
```

Service principal user authentication

To use service principal user authentication with Entra ID:

- Set the AuthenticationMethod property to specify a value of `ActiveDirectoryServicePrincipal`.

- Set the ActiveDirectoryPrincipalID property to specify the Entra ID Identity (PrincipalID) of the Azure SQL logical server.
- Set the ActiveDirectoryPrincipalSecret property to specify the client secret for your Entra ID application.
- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Entra ID server. For example, `myserver.database.windows.net`.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

Note: If the HostNameInCertificate is not specified, the driver automatically uses the value of the ServerName from the URL as the value for validating the certificate.

For example, the following is a connection string with only the required options for making a connection using Entra ID authentication.

```
Connection conn = DriverManager.getConnection
        ("jdbc:datadirect:sqlserver://your_server.database.windows:1433;
        AuthenticationMethod=ActiveDirectoryServicePrincipal;
        ActiveDirectoryPrincipalID=789f8b4c-7a4a-445d-6oe9-7bec14625645;
        ActiveDirectoryPrincipalSecret=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO=");
```

Managed identity authentication

The driver supports the following methods of managed identity authentication:

- *User-assigned authentication*: The driver authenticates using the client ID of the user assigned managed identity.

  To use user-assigned managed identity authentication with Entra ID:

  - Set the AuthenticationMethod property to the value `ActiveDirectoryManagedIdentity`.
  - Set the User property to specify your Microsoft Entra ID (Azure) client ID of the user assigned managed identity.

    Note: The `User` property is optional if only one user-assigned managed identity is configured. This property is required if there are more than one user-assigned managed identities configured.

  - Specify values for minimum required properties for establishing a connection.
    - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Entra ID (Azure) server. For example, `myserver.database.windows.net`.
    - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
  - Optionally, set the DatabaseName property to the database to which you want to connect.

  For example, the following is a connection string with the minimal options for making a connection using managed identity authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:sqlserver://your_server.database.windows:1433;DatabaseName=exampleDB;
AuthenticationMethod=ActiveDirectoryManagedIdentity;User=f4f42d67-6789-4d58-a789-83a3dc123456");
```

- *System-assigned authentication*: The driver authenticates using the identity associated with Azure resources like virtual machines.

  To use system-assigned managed identity authentication with Entra ID:

  - Set the AuthenticationMethod property to specify a value of `ActiveDirectoryManagedIdentity`.
  - Specify values for minimum required properties for establishing a connection.
    - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, `myserver.database.windows.net`.
    - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
  - Optionally, set the DatabaseName property to the database to which you want to connect.

  For example, the following is a connection string with the minimal options for making a connection using managed identity authentication.

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sqlserver://your_server.database.windows:1433;DatabaseName=exampleDB;

   AuthenticationMethod=ActiveDirectoryManagedIdentity");
```

See also

AuthenticationMethod on page 125
User on page 182
Password on page 163
ServerName on page 171
PortNumber on page 164
ActiveDirectoryPrincipalID on page 115
ActiveDirectoryPrincipalSecret on page 116

Kerberos authentication

Your Kerberos environment should be fully configured before you configure the driver for Kerberos authentication. You should refer to your SQL Server documentation and Java documentation for instructions on configuring Kerberos. For a Windows Active Directory implementation, you should also consult your Windows documentation. For a non-Entra ID implementation (on a Windows or non-Windows operating system), you should consult MIT Kerberos documentation.

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

Once your Kerberos environment has been configured, take the following steps to configure the driver.

1. Use one of the following methods to integrate the JAAS configuration file into your Kerberos environment. (See "The JAAS login configuration file" for details.)

   Note: The *install_dir*`/lib/JDBCDriverLogin.conf` file is the JAAS login configuration file installed with the driver. You can use this file or another file as your JAAS login configuration file.

   Note: Regardless of operating system, forward slashes must be used when designating the path of the JAAS login configuration file.

   - Specify a login configuration file directly in your application with the `java.security.auth.login.config` system property. For example:

     ```
     System.setProperty("java.security.auth.login.config","install_dir/lib/JDBCDriverLogin.conf");
     ```

   - Set up a default configuration. Modify the Java security properties file to indicate the URL of the login configuration file with the `login.config.url.`*n* property where *n* is an integer connoting separate, consecutive login configuration files. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration.

a) Open the Java security properties file. The security properties file is the `java.security` file in the `/jre/lib/security` directory of your Java installation.

b) Find the line `# Default login configuration file` in the security properties file.

c) Below the `# Default login configuration file` line, add the URL of the login configuration file as the value for a `login.config.url.`*n* property. For example:

```
# Default login configuration file
login.config.url.1=file:${user.home}/.java.login.config
login.config.url.2=file:install_dir/lib/JDBCDriverLogin.conf
```

2. Ensure your JAAS login configuration file includes an entry with authentication technology that the driver can use to establish a Kerberos connection. (See "The JAAS login configuration file" for details.)

Note: The JAAS login configuration file installed with the driver (*install_dir*`/lib/JDBCDriverLogin.conf`) includes a default entry with the name `JDBC_DRIVER_01`. This entry specifies the Kerberos authentication technology used with an Oracle JVM.

The following examples show that the authentication technology used in a Kerberos environment depends on your JVM.

Oracle JVM

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

IBM JVM

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

3. 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.

  ---

4. Set the driver's AuthenticationMethod connection property to `auto` or `kerberos`. (See "AuthenticationMethod" for details.)

Note: If your are configuring your environment for Kerberos Constrained Delegation (also known as impersonation), AuthenticationMethod must be set to `kerberos`.

5. Optionally, set the ServicePrincipalName connection property if the default value built by the driver does not match the service principal name registered with the KDC.

   By default, the driver builds the ServicePrincipalName by concatenating the service name `MSSQLSvc`, the fully qualified domain name (FQDN) as specified with the ServerName property, the port number as specified with the PortNumber property, and the default realm name as specified in the Kerberos configuration file (`krb5.conf`). If this value does not match the service principal name registered with the KDC, then the value of the service principal name registered with the KDC should be specified for the ServicePrincipalName property.

   The ServicePrincipalName takes the following form.

   `Service_Name/Fully_Qualified_Domain_Name:Port_Number@REALM_NAME`

   See "ServicePrincipalName" for details on the composition of the service principal name.

6. Optionally, set the LoginConfigName connection property if the name of the JAAS login configuration file entry is different from the driver default `JDBC_DRIVER_01`. (See "The JAAS login configuration file" and "LoginConfigName" for details.)

   `JDBC_DRIVER_01` is the default entry name for the JAAS login configuration file (`JDBCDriverLogin.conf`) installed with the driver. When configuring your Kerberos environment, your network or system administrator may have used a different entry name. Check with your administrator to verify the correct entry name.

7. Optionally, set the GSSCredential connection property for Kerberos constrained delegation (sometimes referred to as impersonation).

   Constrained delegation is a Kerberos mechanism that allows a client application to delegate authentication to a second service. See "Constrained delegation" for additional steps to configure your environment.

   AuthenticationMethod must be set to `kerberos` to use constrained delegation.

See also

Kerberos authentication requirements on page 56
The JAAS login configuration file on page 57
Constrained delegation on page 58
AuthenticationMethod on page 125
ServicePrincipalName on page 172
LoginConfigName on page 156
GSSCredential on page 150

Kerberos authentication requirements

Verify that your environment meets the requirements listed in the following table before you configure the driver for Kerberos authentication.

Note: For Windows Entra ID, the domain controller must administer both the database server and the client.

Table 2: Kerberos configuration requirements

Component	Requirements
Database server	No restrictions
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 key distribution center (KDC). Network authentication must be provided by one of the following methods. • Windows Entra ID on SQL Server • MIT Kerberos 1.5 or higher
Client	Java SE 8 or higher must be installed.

See also

Kerberos authentication on page 54

The JAAS login configuration file

The Java Authentication and Authorization Service (JAAS) login configuration file contains one or more entries that specify authentication technologies to be used by applications. To establish Kerberos connections with the driver, the JAAS login configuration file must include an entry specifically for the driver. In addition, the login configuration file must be referenced either by setting the `java.security.auth.login.config` system property or by setting up a default configuration using the Java security properties file.

Setting up a default configuration

To set up a default configuration, you must modify the Java security properties file to indicate the URL of the login configuration file with the `login.config.url.`*n* property where *n* is an integer connoting separate, consecutive login configuration files. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration. The following steps summarize how to modify the security properties file.

1. Open the Java security properties file. The security properties file is the `java.security` file in the `/jre/lib/security` directory of your Java installation.
2. Find the line `# Default login configuration file` in the security properties file.
3. Below the `# Default login configuration file` line, add the URL of the login configuration file as the value for a `login.config.url.`*n* property. For example:

```
# Default login configuration file
login.config.url.1=file:${user.home}/.java.login.config
login.config.url.2=file:install_dir/lib/JDBCDriverLogin.conf
```

JAAS login configuration file entry for the driver

You can create your own JAAS login configuration file, or you can use the `JDBCDriverLogin.conf` file installed in the `/lib` directory of the product installation directory. In either case, the login configuration file must include an entry that specifies the Kerberos authentication technology to be used by the driver.

JAAS login configuration file entries begin with an entry name followed by one or more LoginModule items. Each LoginModule item contains information that is passed to the LoginModule. A login configuration file entry takes the following form.

```
entry_name {
  login_module flag_value module_options
};
```

where:

entry_name

is the name of the login configuration file entry. The driver's LoginConfigName connection property can be used to specify the name of this entry. `JDBC_DRIVER_01` is the default entry name for the `JDBCDriverLogin.conf` file installed with the driver.

login_module

is the fully qualified class name of the authentication technology used with the driver.

flag_value

specifies whether the success of the module is `required`, `requisite`, `sufficient`, or `optional`.

module_options

specifies available options for the LoginModule. These options vary depending on the LoginModule being used.

The following examples show that the LoginModule used for a Kerberos implementation depends on your JVM.

Oracle JVM

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

IBM JVM

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

Refer to Java Authentication and Authorization Service documentation for information about the JAAS login configuration file and implementing authentication technologies.

See also

Kerberos authentication on page 54
LoginConfigName on page 156

Constrained delegation

Constrained delegation is a Kerberos mechanism that allows a client application to delegate authentication to a second service. The client application informs the KDC that the second service is authorized to act on behalf of a specified Kerberos security principal, such as a user that has an Entra ID account. The second service can then delegate authentication to a database service principal name. (Refer to the Microsoft TechNet page About Kerberos constrained delegation for further details.)

To enable constrained delegation:

Important: Before you start, in the `[libdefaults]` section of the `krb5.conf` file, set the `forwardable` flag to `true`.

1. Authenticate the service principal and get a subject from the login context. The service principal needs a Kerberos granting ticket to be authenticated. You can use either a ticket cache or keytab file for the authentication step. The section you define in your JAAS login configuration file determines which method is used for authentication.
2. Call the impersonate method to generate the service ticket for the database user on behalf of the service principal identity.
3. Using the driver's GSSCredential property, specify the GSSCredential generated in the previous steps.
4. Call the driver's connect() method using the Properties object. The Properties object must contain the GSSCredential property and any additional properties needed to establish a connection to the database.

The following example code shows how a GSS credential object can be integrated into a client application to support constrained delegation.

```
    Subject serviceSubject;
    GSSCredential creds;

 //Authenticate the service principal and get a subject from the login context.
    LoginContext lc = new LoginContext("entry_from_your_jaas_config");
    lc.login();
    serviceSubject = lc.getSubject();

 //Call the impersonate method to generate the service ticket for database user
 //on behalf of the service principal identity.
    try {
     creds = Subject.doAs(serviceSubject, new
             PrivilegedExceptionAction<GSSCredential>() {
               public GSSCredential run() throws Exception {
                 GSSManager manager = GSSManager.getInstance();
                 if (serviceCredentials == null) {
              serviceCredentials =
                         manager.createCredential(GSSCredential.INITIATE_ONLY);
               }
               GSSName other = manager.createName("userToImpersonate",
                               GSSName.NT_USER_NAME);
                               return
               ((ExtendedGSSCredential)serviceCredentials).impersonate(other);
             }
            });
    } catch (PrivilegedActionException pae) {
       throw pae.getException();
    }

    final Properties sqlserverProperties;

    sqlserverProperties = new Properties();
 // Set the driver's GSSCredential property and the rest of the database
 // connection properties
    sqlserverProperties.put("GSSCredential", creds);
    sqlserverProperties.put("ServerName", "yourServer");
    sqlserverProperties.put("portNumber", "1433");
    sqlserverProperties.put("authenticationMethod", "Kerberos");
    sqlserverProperties.put("databaseName", "yourDatabase");


    Connection con = DriverManager.getConnection("jdbc:datadirect:sqlserver:",
                     sqlserverProperties);
    DatabaseMetaData  dbmd = con.getMetaData();

    System.out.println( "\nConnected with " + dbmd.getDriverName() + "\n"
```

```
+ " to " + dbmd.getDatabaseProductName() + "\n"
+ " " + dbmd.getDatabaseProductVersion() + "\n"
+ " " + dbmd.getDriverVersion() + "\n");
```

NTLM authentication

How you configure the driver for NTLM depends on your client's operating system:

- On Windows, you can configure the driver for NTLM authentication using either of the following methods:
  - Configuring NTLM authentication without specifying user credentials (Windows Only) on page 60
  - Configuring NTLM authentication by specifying user credentials on page 61
- On UNIX/Linux, configuring the driver for NTLM authentication requires that you specify user credentials. See Configuring NTLM authentication by specifying user credentials on page 61 for details.

NTLM authentication requirements

Verify that your environment meets the requirements listed in the following table before you configure your environment for NTLM authentication.

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

Table 3: NTLM authentication requirements

Component	Requirements
Database server	The server must be running Microsoft SQL Server 2005 or higher.
Domain controller	Network authentication must be provided by NTLM on Microsoft SQL Server 2005 or higher.
Client	Java SE 8 or higher must be installed.

Configuring NTLM authentication without specifying user credentials (Windows Only)

On Windows, you can configure the driver for NTLM authentication such that user credentials do not have to be specified.

To use NTLM authentication without specifying user credentials, the driver must load an NTLM authentication DLL. The product provides the following NTLM authentication DLLs (where *xx* is a 2-digit number):

- DDJDBCAuth*xx*.dll (32-bit)
- DDJDBC64Auth*xx*.dll (Itanium 64-bit)
- DDJDBCx64Auth*xx*.dll (AMD64 and Intel EM64T 64-bit)

The DLLs are located in the *install_dir*/lib directory (where *install_dir* is your product installation directory). If the application using NTLM authentication is running in a 32-bit JVM, the driver automatically uses DDJDBCAuth*xx*.dll. Similarly, if the application is running in a 64-bit JVM, the driver uses DDJDBC64Auth*xx*.dll or DDJDBCx64Auth*xx*.dll.

To configure the driver:

1. Set the AuthenticationMethod property to `auto` (default) or `ntlm`.
2. By default, the driver looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install the driver in a directory that is not on the Windows system path, perform one of the following actions to ensure the driver can load the DLLs:
   - Add the *install_dir*`/lib` directory to the Windows system path, where *install_dir* is your product installation directory.
   - Copy the NTLM authentication DLLs from *install_dir*`/lib` to a directory that is on the Windows system path, where *install_dir* is your product installation directory.
   - Set the LoadLibraryPath property to specify the location of the NTLM authentication DLLs. For example, if you install the driver in a directory named "DataDirect" that is not on the Windows system path, you can use the LoadLibraryPath property to specify the directory containing the NTLM authentication DLLs: `jdbc:datadirect:sqlserver://server3:1521;`
   - `DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;`
   - `User=test;Password=secret`
3. If using NTLM authentication with a Security Manager on a Java Platform, security permissions must be granted to allow the driver to establish connections.

Configuring NTLM authentication by specifying user credentials

You can configure the driver for NTLM authentication with the specification of user credentials.

To configure the driver:

1. Set the AuthenticationMethod property to either of the following values.
   - `ntlmjava` to use NTLMv1 or NTLMv2 depending on the size of the password. NTLMv1 is used if the password is 14 bytes or less; NTLMv2 is used if the password is more than 14 bytes.
   - `ntlm2java` to use NTLMv2 protocols to connect to a server that is restricted to using NTLMv2 authentication.

   Note: See AuthenticationMethod on page 125 for more information on setting the AuthenticationMethod property.
2. Set the Domain property to provide the name of the domain server that administers the database.

   Note: Alternatively, you can set the domain server name using the User property.
3. Set the User property to provide the user ID.
4. Set the Password property to provide the password.
5. If using NTLM authentication with a Security Manager on a Java Platform, security permissions must be granted to allow the driver to establish connections.

TLS/SSL encryption

The driver supports SSL encryption. Depending on your Microsoft SQL Server configuration, you can choose to encrypt all data, including the login request, or encrypt the login request only. Encrypting login requests, but not data, is useful for the following scenarios:

- When your application needs security, but cannot afford to pay the performance penalty for encrypting data transferred between the driver and server.
- When the server is not configured for SSL, but your application still requires a minimum degree of security.

Note: When SSL is enabled, the driver communicates with database protocol packets set by the server's default packet size. Any value set by the PacketSize property is ignored.

Using TLS/SSL with Microsoft SQL Server

If your Microsoft SQL Server database server has been configured with an TLS/SSL certificate signed by a trusted CA, the server can be configured so that TLS/SSL encryption is either optional or required. When required, connections from clients that do support TLS/SSL encryption fail.

Although a signed trusted TLS/SSL certificate is recommended for the best degree of security, Microsoft SQL Server can provide limited security protection even if a TLS/SSL certificate has not been configured on the server. If a trusted certificate is not installed, the server will use a self-signed certificate to encrypt the login request, but not the data.

The following table shows how the different EncryptionMethod property values behave with different Microsoft SQL Server configurations.

Table 4: EncryptionMethod property values and Microsoft SQL Server configurations

Value	No TLS/SSL Certificate	TLS/SSL Certificate	
		TLS/SSL Optional	TLS/SSL Required
noEncryption	Login request and data are not encrypted.	Login request and data are not encrypted.	Connection attempt fails.
SSL	Connection attempt fails.	Login request and data are encrypted.	Login request and data are encrypted.
Strict	Connection attempt fails.	Login request and data are encrypted.	Login request and data are encrypted.
requestSSL	Login request and data are not encrypted.	Login request and data are encrypted.	Login request and data are encrypted.
loginSSL	Login request is encrypted, but data is not encrypted	Login request is encrypted, but data is not encrypted.	Login request and data are encrypted.

Configuring TLS/SSL encryption

Take the following steps 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.

- Choose the type of encryption for your application:
  - If you want the driver to encrypt all data, including the login request, set the EncryptionMethod property to one of the following:
    - 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.
    - requestSSL: Data is encrypted using TLS/SSL. If the database server does not support TLS/SSL, the driver establishes an unencrypted connection.
    - Strict: The driver uses the TDS (Tabular Data Stream) 8.0 protocol to support TLSv1.3 encryption for SQL Server connections. You must specify this value when your server is configured with `Force Strict Encryption=yes`.

      Important: When using strict connection encryption:

      - The driver validates the certificates sent by the server (`ValidateServerCertificate=true`) for the connection, regardless of the setting of the ValidateServerCertificate property.
      - You must specify a truststore containing the server certificate against which the server will be validated at connection.

  - If you want the driver to encrypt only the login request, set the EncryptionMethod property to loginSSL.
- Use the CryptoProtocolVersion property to specify acceptable cryptographic protocol versions (for example, TLSv1.2) supported by your server.

  Note: TLSv1.3 is currently supported only when strict connection encryption is enabled (`EncryptionMethod=Strict`).

- Specify the location and password of the truststore file used for TLS/SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- To validate certificates sent by the database server, set the ValidateServerCertificate property to `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.

Always Encrypted

Microsoft supports the Always Encrypted feature for Azure SQL Databases and SQL Server databases beginning with SQL Server 2016. Always Encrypted functionality provides improved security by storing sensitive data on the server in an encrypted state. When sensitive data is queried by the application, the driver transparently decrypts data from encrypted columns and returns them to the application. Conversely, when encrypted data needs to be passed to the server, the driver transparently encrypts parameter values before sending them for storage. As a result, sensitive data is visible only to authorized users of the application, not by those who maintain the data. This reduces exposure to a number of potential vulnerabilities, including server-side security breaches and access by database administrators who would not otherwise be authorized to view the data.

Note: Always Encrypted is not supported in FIPS enabled environments.

Always Encrypted functionality employs a column master key and column encryption key to process encrypted data. The column encryption key is used to encrypt sensitive data in an encrypted column, while the column master key is used to encrypt column encryption keys. To prevent server-side access to encrypted data, the column master key is stored in a keystore that is separate from the server that contains the data. When Always Encrypted is enabled, the driver uses the steps described in this section to retrieve keys and negotiate the decryption of encrypted data.

By design, data stored in encrypted columns cannot be accessed without first being retrieved and decrypted by the driver. Although this restriction improves security, it also prevents literal values within these columns to be referenced when issuing a statement. As a result, statements can only reference encrypted columns using parameter markers.

When the application executes a parameterized statement with Always Encrypted enabled:

1. The driver executes a stored procedure to determine from the server whether there are any encrypted columns referenced by the statement.
2. If any columns are encrypted, the driver retrieves the encrypted column metadata, encrypted column encryption key, and the location of the column master key for each parameter to be encrypted.
3. The driver retrieves the column master key from the keystore; then, uses it to decrypt the column encryption key. After decryption, the column encryption key is cached in a decrypted state for subsequent operations or discarded. See the "Using Keystore Providers" section for more information.

   Note: You can dictate the length of time column encryption keys are persisted in the cache using the AEKeyCacheTTL property. See "Caching column encryption keys" for more information.

4. The driver encrypts the parameters using the decrypted column encryption key.
5. The driver sends the statement with encrypted values to the server for processing.
6. If applicable, the server returns the result set, along with the encryption algorithm information, encrypted column encryption key, and location of the column master keys.
7. If the column encryption key is not cached, the driver retrieves column master key from the keystore; then uses it to decrypt the column encryption key.
8. The driver decrypts the result set using the column encryption key and returns it to the application.

See "Enabling Always Encrypted" for information on configuring Always Encrypted with the driver.

See also

Using keystore providers on page 65
Caching column encryption keys on page 66

Enabling Always Encrypted on page 65

Enabling Always Encrypted

You can configure Always Encrypted behavior for the driver by specifying the following values for the ColumnEncryption connection property:

- If set to `Enabled`, the driver fully supports Always Encrypted functionality. The driver transparently decrypts result sets and returns them to the application. In addition, the driver transparently encrypts parameter values that are associated with encrypted columns.
- If set to `ResultsetOnly`, only decryption is enabled. The driver transparently decrypts result sets and returns them to the application. Queries containing parameters that affect encrypted columns will return an error.
- If set to `Disabled` (the default), Always Encrypted functionality is disabled. The driver does not attempt to decrypt data from encrypted columns and returns the data as binary-formatted cipher text. However, statements containing parameters that reference encrypted columns are not supported and will return an error.

By default, Always Encrypted is disabled (`ColumnEncryption=Disabled`). For more information on configuring the ColumnEncryption property, see "ColumnEncryption."

When Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`), the driver must be configured to use a keystore provider; otherwise, an error will be returned. See "Using keystore providers" for details.

See also

Using keystore providers on page 65
ColumnEncryption on page 132

Using keystore providers

Keystore providers securely store the column master keys used for decrypting the column encryption keys employed by Always Encrypted functionality. The driver requires that a keystore provider be used when always encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). The following section describes how to configure the driver to use the supported keystore providers.

Azure Key Vault

The Azure Key Vault is a certificate repository hosted on Azure platforms. Using Azure Key Vault offers several advantages, including the ability for applications on any platform to access keys. In addition, since the keys are centrally stored, they do not need to be copied to and cached on a local machine. However, unless your application is running on Azure, calls to the key vault must be made over a WAN, which can negatively impact performance. To use Azure Key Vault, values for the following properties must be provided:

- AEKeystorePrincipalId: Specifies the principal ID for the Azure Key Vault. The principal ID is the Application ID created during Azure App Registration. See "KeyStorePrincipalId" for a detailed description.
- AEKeystoreClientSecret: Specifies the Client Secret used to access the Azure Key Vault. See "AEKeystoreClientSecret" for a detailed description.

Java KeyStore

Java Keystore is a repository of certificates for Java platforms. Similar to Azure Key Vault, the column master key is stored centrally, which means keys do not need to be cached on local machines. However, unlike Azure Key Vault, access to the Java Keystore is limited to applications running on Java platforms. To use Java Keystore, values for the following properties must be provided:

- AEKeystoreLocation: Specifies the absolute path to the Java KeyStore file. See "AEKeystoreLocation" for details.
- AEKeystoreSecret: Specifies the password used to access the Java KeyStore file. See "AEKeystoreSecret" for details.

See also

ColumnEncryption on page 132
AEKeystorePrincipalId on page 120
AEKeystoreClientSecret on page 118
AEKeystoreLocation on page 119
AEKeystoreSecret on page 121

Caching column encryption keys

Caching column encryption keys improves performance by reducing the overhead associated with fetching and decrypting the keys for the same data multiple times. For security purposes, the driver empties keys from the cache when a connection closes; however, for applications that remain connected for long periods of time, you may want to delete the keys before the connection ends. You can determine the length of time the driver caches keys by specifying the following values for the AEKeyCacheTTL property:

- If set to `-1`, the driver caches column encryption keys for the life of the connection. The keys are deleted when the connection is closed or added to the connection pool.
- If set to `0`, the driver does not cache column encryption keys.
- If set to *x*, the driver caches column encryption keys for the specified number of seconds before deleting them. The timer starts for a key when it is first accessed and added to the cache. The timer does not reset if you access it after it has been added to the cache. The keys are deleted when the timer expires, or the connection is closed or added to the connection pool.

By default, the driver caches keys for `7200` seconds. See "AEKeyCacheTTL" for details.

See also

AEKeyCacheTTL on page 117

Enabling parameter metadata discovery

The driver initiates the processing of encrypted parameters by passing the T-SQL for a prepared statement to the sp_describe_parameter_encryption system stored procedure, which is then used to return metadata for the encrypted parameters in statement. To maintain data type integrity, the server requires that the T-SQL data type, length, precision and scale values specified by the driver match those of the underlying native data type referenced by the parameters in the T-SQL statement. However, since some of the native data types do not have a one-to-one mapping to a JDBC type[8], the driver may not be able to communicate the T-SQL type to the server when calling this procedure. When this occurs, the procedure will fail to execute with an `Operand type clash` error.

[8] For example., because there is no SQL_MONEY type for JDBC, the native Money and Decimal types both map to SQL_DECIMAL

To correct this issue, functionality was added to allow the driver to automatically discover the underlying type and adjust the T-SQL passed to the the sp_describe_parameter_encryption procedure. This behavior is disabled by default when Always Encrypted functionality is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`). To enable data type discovery, configure the following connection properties with the values provided:

```
DescribeInputParameters=DESCRIBEALL
DescribeOutputParameters=DESCRIBEALL
```

When these values are specified, the driver makes an extra call to the server to retrieve accurate metadata to pass to the sp_describe_column_encryption procedure, which results in the gathering of encryption metadata and allows for encryption and decryption to succeed.

See also

DescribeInputParameters on page 139
DescribeOutputParameters on page 140

Connection string example

The following connection strings configure the driver to use Always Encrypted with the minimum required properties.

For connections using Azure Key Vault:

```
"jdbc:datadirect:sqlserver://MyServer:1433;
      AEKeystorePrincipalId=789f8b4c-7a4a-445d-6oe9-7bec14625645;
      AEKeyStoreClientSecret=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO=;
      ColumnEncryption=Enabled;"
```

For connections Java KeyStore:

```
"jdbc:datadirect:sqlserver://MyServer:1433;
      AEKeystoreLocation=/usr/java/jre/lib/security/cacerts;AEKeystoreSecret=secret;
      ColumnEncryption=Enabled;"
```

See also

AEKeystorePrincipalId on page 120
AEKeystoreClientSecret on page 118
ColumnEncryption on page 132
AEKeystoreLocation on page 119
AEKeystoreSecret on page 121
Connection property descriptions on page 103

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 SQL Server are not FIPS compliant, the Always Encrypted feature is not supported in a FIPS enabled environment. See "Always Encrypted" for more information about the feature.

See also

Always Encrypted on page 64

Connecting to named instances

Microsoft SQL Server supports multiple instances of a database running concurrently on the same server. An instance is identified by an instance name. To connect to a named instance using a connection URL, use the following URL format.

```
jdbc:datadirect:sqlserver://server_name\\instance_name
```

Note: The first backslash character (\) in *instance_name* is an escape character.

where:

server_name

is the IP address or hostname of the server.

instance_name

is the name of the instance to which you want to connect on the server.

For example, the following connection URL connects to an instance named instance1 on server1.

```
jdbc:datadirect:sqlserver://server1\\instance1;User=test;Password=secret
```

To connect to a named instance using a data source, you specify the ServerName property. For example:

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1\\instance1");
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
```

Azure Synapse Analytics and Analytics Platform System

The driver transparently connects to Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System; however, the following limitations to features and functionality apply.

- No support for connecting to an instance using an IP address for the server. A named instance must be specified for the ServerName property.
- No support for unquoted identifiers. The driver always enforces ANSI rules regarding quotation marks for Azure Synapse Analytics and Analytics Platform System connections.
- No support for connection pooling reauthentication.
- No support for Data Definition Language (DDL) queries within transactions.
- No support for closing holdable cursors when a transaction is committed.
- No support for server-side cursors; therefore:
  - Scroll-sensitive result sets are not supported.
  - The UseServerSideUpdatableCursors property is set to false, and server-side cursors are not used.
- No support for XA connections.
- Support for isolation levels is limited to only the read uncommitted level. See "Isolation Levels" for more information.
- No support for the following SQL Server data types in either the Azure Synapse Analytics or Analytics Platform System.

decimal() identity	timestamp
image	tinyint identity
numeric() identity	ntext
smallint identity	xml
text	

- Support for scalar string functions is limited to the following functions.

ASCII	LEFT	RTRIM
CHAR	LTRIM	SOUNDEX
CONCAT	REPLACE	SPACE
DIFFERENCE	RIGHT	SUBSTRING

- Support for scalar numeric functions is limited to the following functions.

ABS	EXP	ROUND
ACOS	FLOOR	SIGN
ASIN	LOG	SIN
ATAN	LOG10	SQRT
CEILING	PI	TAN
COS	POWER	TRUNCATE
COT	RADIANS	
DEGREES	RAND (Azure Warehouse only)	

- Support for scalar date and time functions is limited to the following functions.

CURDATE	DAYOFWEEK	QUARTER
CURRENT_DATE	DAYOFYEAR	SECOND
CURRENT_TIME	HOUR	WEEK
CURTIME	MINUTE	YEAR
DAYNAME	MONTH	
DAYOFMONTH	MONTHNAME	

Note:

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

See also

Data types on page 17

Support for Microsoft Fabric

The driver provides read-write access to the Synapse Data Warehouse endpoints of Microsoft Fabric; however, the following limitations to features and functionality apply:

Note: The Microsoft Fabric data source is constantly evolving; therefore, the limitations given in this section may not apply in the future. For the latest information, refer to the Microsoft Fabric documentation.

- The PRIMARY KEY and UNIQUE constraints are only supported when NONCLUSTERED and NOT ENFORCED are both used, whereas the FOREIGN KEY constraint is only supported when NOT ENFORCED is used. These keys cannot be created while creating a table, but can be added later using ALTER TABLE.
- No support for distributed transactions and named transactions.
- No support for CREATE INDEX.
- `getTablePrivileges()` and `getCoumnPrivileges()` return empty result sets.
- `connection.getMetaData().getSchemas()` returns the schema names in an unsorted order.
- CREATE FUNCTION can create inline table-value functions, but not scalar functions.
- No support for the following T-SQL commands, queries, and features:

• ALTER TABLE (ADD, ALTER, and DROP COLUMN): Currently, only the following subsets of ALTER TABLE are supported in Warehouse in Microsoft Fabric:	• MERGE	• Schema and table names containing / or \

• ADD nullable columns of supported column data types. • ADD or DROP PRIMARY KEY, UNIQUE, and FOREIGN KEY column constraints, but only if the NOT ENFORCED option has been specified. All other ALTER TABLE operations are blocked.		
• CREATE ROLE	• OPENROWSET	• SELECT - FOR XML
• CREATE USER	• PREDICT	• SET ROWCOUNT
• Hints	• Queries targeting system and user tables	• SET TRANSACTION ISOLATION LEVEL
• IDENTITY columns	• Recursive queries	• Temporary tables
• Manually created multi-column statistics	• smallint identity	• Triggers
• Materialized views	• Result Set Caching	

- No support for the following keywords:

• ALLOW_SNAPSHOT_ISOLATION	• COLLATE
• COLUMN ENCRYPTION	• DECLARE CURSOR

- No support for the following SQL Server data types:

• bigint identity	• money	• smallmoney
• binary	• nchar	• text
• datetime	• ntext	• timestamp
• datetimeoffset	• numeric() identity	• tinyint
• decimal() identity	• nvarchar	• tinyint identity
• image	• nvarchar(max)	• uniqueidentifier
• int identity	• smallint identity	

- Support for scalar string functions is limited to the following functions:

• ASCII	• LEFT	• RTRIM
• CHAR	• LTRIM	• SOUNDEX
• CONCAT	• REPLACE	• SPACE
• DIFFERENCE	• RIGHT	• SUBSTRING

- Support for scalar numeric functions is limited to the following functions:

• ABS	• EXP	• ROUND
• ACOS	• FLOOR	• SIGN
• ASIN	• LOG	• SIN
• ATAN	• LOG10	• SQRT
• CEILING	• PI	• TAN
• COS	• POWER	• TRUNCATE
• COT (ADW only)	• RADIANS	
• DEGREES	• RAND	

- Support for scalar date and time functions is limited to the following functions:

• CURDATE	• DAYOFWEEK	• QUARTER
• CURRENT_DATE	• DAYOFYEAR	• SECOND
• CURRENT_TIME	• HOUR	• WEEK
• CURTIME	• MINUTE	• YEAR
• DAYNAME	• MONTH	
• DAYOFMONTH	• MONTHNAME	

Note:

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

See also

Data types on page 17

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. In the following example, the server name MyServer can resolve to either type of address:

```
jdbc:datadirect:sqlserver://MyServer:1433;
   DatabaseName=Test;User=admin;Password=secret
```

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

```
jdbc:datadirect:sqlserver://123.456.78.90:1433;
   DatabaseName=MyDB;User=admin;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 an IPv6 address:

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServer DataSource");
mds.setServerName("2001:DB8:0:0:8:800:200C:417A");
...
```

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:sqlserver://[2001:DB8:0:0:8:800:200C:417A]:1433;
   DatabaseName=MyDB;User=admin;Password=secret
```

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

```
jdbc:datadirect:sqlserver://[0000:0000:0000:0000:0000:FFFF:123.456.78.90]:1433;
   DatabaseName=MyDB;User=admin;Password=secret
```

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

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

Failover

Take the following steps to configure failover.

1. Specify the primary and alternate servers:
   - Specify your primary server using a connection URL or data source.
   - Specify one or multiple alternate servers by setting the AlternateServers property.

   See Specifying primary and alternate servers on page 74.

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

Note: If using failover with Microsoft Cluster Server (MSCS), which determines the alternate server for failover instead of the driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;
  DatabaseName=TEST;User=test;Password=secret;
  AlternateServers=(server1:1433;DatabaseName=TEST)
```

2. Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (FailoverMode=`connect`).
3. If FailoverMode=`extended` or FailoverMode=`select`, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=`nonAtomic`).
4. Optionally, configure the connection retry feature. See Specifying connection retry on page 76.
5. 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`).

Specifying primary and alternate servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the SQL Server driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,
server3:1433;DatabaseName=TEST3)
```

In this example:

```
...server1:1433;DatabaseName=TEST...
```

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=(server2:1433;DatabaseName=TEST2,server3:1433;
DatabaseName=TEST3)
```

Similarly, the same connection information for the primary and alternate servers specified using a JDBC data source would look like this:

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1");
mds.setPortNumber(1433);
mds.setDatabaseName("TEST");
mds.setUser("test");
```

```
mds.setPassword("secret");
mds.setAlternateServers("server2:1433;DatabaseName=TEST2, server3:1433;
   DatabaseName=TEST3")
```

In this example, connection information for the primary server is specified using the ServerName, PortNumber, and DatabaseName properties. Connection information for alternate servers is specified using the AlternateServers property.

The SQL Server driver also allows you to specify connections to named instances, multiple instances of a Microsoft SQL Server database running concurrently on the same server. If specifying named instances for the primary and alternate servers, the connection URL would look like this:

```
jdbc:datadirect:sqlserver://server1\\instance1;User=test;Password=secret;
AlternateServers=(server2\\instance2:1433;DatabaseName=TEST2,
server3\\instance3:1433;DatabaseName=TEST3)
```

Similarly, the same connection information to named instances for the primary and alternate servers specified using a JDBC data source would look like this:

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1\\instance1");
mds.setPortNumber(1433);
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
mds.setAlternateServers("server2\\instance2:1433;DatabaseName=
   TEST2,server3\\instance3:1433;DatabaseName=TEST3")
```

To connect to a named instance using a data source, you specify the named instance on the primary server using the ServerName property.

See Connecting to named instances on page 68 for more information about connecting to named instances on Microsoft SQL Server.

The value of the AlternateServers property is a string that has the format:

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

or, if connecting to named instances:

```
(servername1\\instance1[;property=value][,servername2\\instance2
[;property=value]]
```

where:

servername1

is the IP address or server name of the first alternate database server, *servername2* is the IP address or server name of the second alternate database server, and so on. The IP address or server name is required for each alternate server entry.

instance1

is the named instance on the first alternate database server, *servername2* is the named instance on the second alternate database server, and so on. If connecting to named instances, the named instance is required for each alternate server entry.

port1

is the port number on which the first alternate database server is listening, *port2* is the port number on which the second alternate database server is listening, and so on. The port number is optional for each alternate server entry. If unspecified, the port number specified for the primary server is used. If a port number is unspecified for the primary server, a default port number of 1433 is used.

property=value

is the DatabaseName connection property. This property is optional for each alternate server entry. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,
server3:1433;DatabaseName=TEST3)
```

or, if connecting to named instances:

```
jdbc:datadirect:sqlserver://server1\\instance1:1433;DatabaseName=TEST;
User=test;Password=secret;AlternateServers=(server2\\instance2:1433;
DatabaseName=TEST2,server3\\instance3:1433;DatabaseName=TEST3)
```

If you do not specify the DatabaseName connection property in an alternate server entry, the connection to that alternate server uses the property specified in the URL for the primary server. For example, if you specify DatabaseName=TEST for the primary server, but do not specify a database name in the alternate server entry as shown in the following URL, the driver tries to connect to the TEST database on the alternate server:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433,server3:1433)
```

Specifying connection retry

Connection retry allows the SQL Server driver to retry connections to the primary database server, and if specified, alternate servers until a successful connection is established. You use the ConnectionRetryCount and ConnectionRetryDelay properties to enable and control how connection retry works. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,
server3:1433;DatabaseName=TEST3);ConnectionRetryCount=2;
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the SQL Server 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.

Configuring failover with Microsoft Cluster Server

Microsoft SQL Server provides Microsoft Cluster Server (MSCS), an advanced database replication technology. The failover functionality provided by the driver does not require this technology, but can work with MSCS to provide comprehensive failover protection. If using failover with MSCS, which determines the alternate server for failover instead of the driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;
  User=test;Password=secret;AlternateServers=(server1:1433;
  DatabaseName=TEST)
```

In addition, alternate servers must mirror data on the primary server or be part of a configuration where multiple database nodes share the same physical data.

Always On Availability Groups

The driver supports Always On Availability Groups. Introduced in SQL Server 2012, Always On Availability Groups is a replica-database environment that provides a high-level of data availability, protection, and recovery. Follow the proceeding guidelines to use the driver with Always On Availability Groups.

- You must specify the virtual network name (VNN) of the availability group listener with the ServerName property to connect to an Always On Availability group.
- Set the ApplicationIntent property to `ReadOnly`. By setting applicationIntent to `ReadOnly` and querying read-only database replicas when possible, you can improve efficiency by reducing the workload on read-write nodes.
- Set the MultiSubnetFailover property to `true`. This allows the driver to attempt parallel connections to all the IP addresses associated with an Availability Group. This offers improved response time over traditional failover, which attempts connections to alternate servers one at a time.

See also

ServerName on page 171
ApplicationIntent on page 123
MultiSubnetFailover on page 160

Performance considerations

You can optimize your application's performance if you set the SQL Server driver connection properties as described in this section:

Always Encrypted: The following options related to the Always Encrypted feature affect performance:

- **ColumnEncryption**: Due to the overhead associated with encrypting and decrypting data, Always Encrypted functionality can adversely affect performance when enabled. If your application does not require access to encrypted columns, you can disable this property (`ColumnEncryption=Disabled`) for improved performance. Alternatively, if your application only needs to retrieve and decrypt columns, not update them, you can improve performance over the behavior of the `Enabled` setting by specifying a value of `ResultsetOnly` for this property. Note that when using this setting, queries containing parameters that affect encrypted columns will return an error.

- **AEKeyCacheTTL**: When Always Encrypted functionality is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`), you can determine how long, in seconds, column encryption keys are cached using the AEKeyCacheTTL property. Caching column encryption keys can provide performance gains by reducing the overhead associated with fetching and decrypting keys for the same data multiple times during a connection. Specifying larger values for this option increases the length of time that a column encryption key persists in the cache; therefore, improving performance in some scenarios. Alternatively, by specifying a value of `-1`, you can configure the driver to persist keys for the life of the connection. Note that column encryption keys are designed to be deleted from the cache as a security measure and should not be stored for long periods of time.

ApplicationIntent: You can shift load away from the read-write nodes of your database cluster to read-only nodes by setting this connection property to `readOnly` and querying read-only database replicas when possible.

EnableBulkLoad: For batch inserts, the driver can use native bulk load protocols instead of the batch mechanism. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. Set this property to `true` to allow existing applications with batch inserts to take advantage of bulk load without requiring changes to the code.

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.

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.

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.

PacketSize: Typically, it is optimal for the client to use the maximum packet size that the server allows. This reduces the total number of round trips required to return data to the client, thus improving performance. Therefore, performance can be improved if this property is set to the maximum packet size of the database server.

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.

SelectMethod: In most cases, using server-side database cursors impacts performance negatively. However, if the following four variables are true in your application, the best setting for this property is cursor, which means use server-side database cursors:

- Your application contains queries that retrieve large amounts of data.
- Your application executes a SQL statement before processing or closing a previous large result set and does this multiple times.
- Large result sets use forward-only cursors.

SnapshotSerializable: Snapshot Isolation provides transaction-level read consistency and an optimistic approach to data modifications by not acquiring locks on data until data is to be modified. This Microsoft SQL Server feature can be useful if you want to consistently return the same result set even if another transaction has changed the data and 1) your application executes many read operations or 2) your application has long running transactions that could potentially block users from reading data. This feature has the potential to eliminate data contention between read operations and update operations. When this connection property is set to `true` (thereby, you are using Snapshot Isolation), performance is improved due to increased concurrency.

UseServerSideUpdatableCursors: In most cases, using server-side updatable cursors improves performance. However, this type of cursor cannot be used with insensitive result sets or with sensitive results sets that are not generated from a database table that contains a primary key.

See Server-side updatable cursors on page 88 for more information about using server-side updatable cursors.

See also

Connection property descriptions on page 103
ApplicationIntent on page 123
ColumnEncryption on page 132
EnableBulkLoad on page 142
EncryptionMethod on page 144
InsensitiveResultSetBufferSize on page 153
LongDataCacheSize on page 158
MaxPooledStatements on page 159
PacketSize on page 162
ResultSetMetaDataOptions on page 170
SelectMethod on page 170
SnapshotSerializable on page 174
UseServerSideUpdatableCursors on page 183
Server-side updatable cursors on page 88

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:

- Returning and inserting/updating XML data
- DML with results
- Parameter metadata support
- ResultSet metadata support
- Snapshot isolation level
- Scrollable cursors
- Server-side updatable cursors
- JTA support: installing stored procedures
- Distributed transaction cleanup
- Large object (LOB) support
- Batch Inserts and Updates
- Rowset support
- Auto-generated keys support
- Null values
- DataDirect Bulk Load

- Inserts into IDENTITY columns for data replication

Returning and inserting/updating XML data

The 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=`longvarchar` or XMLDescribeType=`longvarbinary`, the driver maps the XML data type to the JDBC LONGVARCHAR or LONGVARBINARY 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 LONGVARCHAR 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 LONGVARBINARY data type, the driver would return XML data as binary data:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=jdbc;User=test;
Password=secret;XMLDescribeType=longvarbinary
```

Returning XML data as character data

When XMLDescribeType=`longvarchar`, the driver returns XML data as character data. The result set column is described with a column type of LONGVARCHAR and the column type name is xml.

When XMLDescribeType=`longvarchar`, 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: This 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.

If XMLDescribeType=`longvarbinary`, 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.

Returning XML data as binary data

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

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.

If XMLDescribeType=`longvarchar`, 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.

Inserting/updating XML as 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 returned by 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.

Inserting/updating XML as 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()
- ResultSet.updateObject()

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

DML with results

The driver supports the Microsoft SQL Server Output clause for Insert, Update, and Delete statements. For example, suppose you created a table with the following statement:

```
CREATE TABLE table1(id int, name varchar(30))
```

The following Update statement updates the values in the id column of table1 and returns a result set that includes the old ID (replaced by the new ID), the new ID, and the name associated with these IDs:

```
UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId, inserted.id as
   newId, inserted.name
```

The driver returns the results of Insert, Update, or Delete statements and the update count in separate result sets. The output result set is returned first, followed by the update count for the Insert, Update, or Delete statement. To execute DML with Results statements in an application, use the Statement.execute() or PreparedStatement.execute() method. Then, use Statement.getMoreResults () to obtain the output result set and the update count. For example:

```
String sql = "UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId, " +
   "inserted.id as newId, inserted.name";
boolean isResultSet = stmt.execute(sql);
int   updateCount = 0;
while (true) {
   if (isResultSet) {
         resultSet = stmt.getResultSet();
         while (resultSet.next()) {
            System.out.println("oldId: " + resultSet.getInt(1) +
                               "newId: " + resultSet.getInt(2) +
                               "name: " + resultSet.getString(3));
         }
         resultSet.close();
   }
   else {
         updateCount = stmt.getUpdateCount();
         if (updateCount == -1) {
            break;
         }
         System.out.println("Update Count: " + updateCount);
   }
   isResultSet = stmt.getMoreResults();
}
```

Parameter metadata support

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

Insert, Update, and Delete statements

The SQL Server driver supports returning parameter metadata for the following forms of Insert, Update, and Delete statements:

- `INSERT INTO foo VALUES (?, ?, ?)`
- `INSERT INTO foo (col1, col2, col3) VALUES (?, ?, ?)`
- `UPDATE foo SET col1=?, col2=?, col3=? WHERE col1 operator ? [{AND | OR} col2 operator ?]`
- `DELETE FROM foo WHERE col1 operator ?`

where *operator* is any of the following SQL operators: =, <, >, <=, >=, and <>.

Select statements

The SQL Server driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

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 SQL Server driver does not support returning parameter metadata for stored procedure arguments.

ResultSet metadata support

If your application requires table name information, the SQL Server driver can return table name information in ResultSet metadata for Select statements. By setting the ResultSetMetaDataOptions property to 1, the SQL Server 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.

When the ResultSetMetaDataOptions property is set to 1 and the ResultSetMetaData.getTableName() method is called, the table name information that is returned by the SQL Server 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 SQL Server 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 SQL Server 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 SQL Server 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 SQL Server 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.

Snapshot isolation level

The driver supports snapshot isolation level.

Note: Snapshot isolation level is not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

You can use snapshot isolation level in either of the following ways:

- Setting the SnapshotSerializable property changes the behavior of the Serializable isolation level to use the Snapshot isolation level. This allows an application to use the Snapshot isolation level with no or minimum code changes. See SnapshotSerializable on page 174 for more information.
- Importing the ExtConstants class allows you to specify the TRANSACTION_SNAPSHOT or TRANSACTION_SERIALIZABLE isolation levels for an individual statement in the same application. The ExtConstants class in the com.ddtek.jdbc.extensions package defines the TRANSACTION_SNAPSHOT

constant. For example, the following code imports the ExtConstants class and sets the TRANSACTION_SNAPSHOT isolation level.

```
import com.ddtek.jdbc.extensions.ExtConstants;
Connection.setTransactionIsolation(
   ExtConstants.TRANSACTION_SNAPSHOT)
```

Scrollable cursors

The SQL Server driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

Server-side updatable cursors

The driver can use client-side cursors or server-side cursors to support updatable result sets.

Note: Server-side updatable cursors are not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

By default, the driver uses client-side cursors because this type of cursor can work with any result set type. Using server-side cursors typically can improve performance, but server-side cursors cannot be used with scroll-insensitive result sets or with scroll-sensitive result sets that are not generated from a database table that contains a primary key. To use server-side cursors, set the UseServerSideUpdatableCursors property to `true`.

When the UseServerSideUpdatableCursors property is set to `true` and a scroll-insensitive updatable result set is requested, the driver downgrades the request to a scroll-insensitive read-only result set. Similarly, when a scroll-sensitive updatable result set is requested and the table from which the result set was generated does not contain a primary key, the driver downgrades the request to a scroll-sensitive read-only result set. In both cases, a warning is generated.

When server-side updatable cursors are used with sensitive result sets that are generated from a database table that contains a primary key, the following changes you make to the result set are visible:

- Own Inserts are visible. Others Inserts are not visible.
- Own and Others Updates are visible.
- Own and Others Deletes are visible.

Using the default behavior of the driver (UseServerSideUpdatableCursors=`false`), those changes are not visible.

See also

UseServerSideUpdatableCursors on page 183

JTA support: installing stored procedures

The driver supports distributed transactions through JTA.

Note: Distributed transactions through JTA are not supported for Microsoft Azure, Microsoft Azure Synapse Analytics, or Microsoft Analytics Platform System.

To use JDBC distributed transactions through JTA, use the following procedure to install Microsoft SQL Server JDBC XA procedures. Repeat this procedure for any Microsoft SQL Server installation that uses distributed transactions.

If you have multiple instances of Microsoft SQL Server on the same machine, you can edit the .sql script file with a text editor to specify a fully qualified path to the sqljdbc.dll file for a particular instance. You will run one of two available script files depending on the version of SQL Server you are using.

- For SQL Server 2008 or higher, the instjdbc.sql script should be used.
- For SQL Server 2005, the instjdbc_2005.sql script should be used.

For example, if you want to install XA Procedures for an instance named "MSSQL.2," modify the .sql script file as shown and run it as described in the following procedure.

```
/*
**  add references for the stored procedures
*/
print 'creating JDBC XA procedures'
go
sp_addextendedproc 'xp_jdbc_open',
   'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_open2',
   'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_close',
   'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_close2',
   'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_start',
   'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
...
```

To install stored procedures for JTA:

1. Stop the Microsoft SQL Server instance.
2. Copy the appropriate 32-bit or 64-bit sqljdbc.dll file to the *SQL_Server_Root*/bin directory of the Microsoft SQL Server database server:

sqljdbc.dll Version	File Location
32-bit	*install_dir*/SQLServer JTA/32-bit
64-bit Itanium	*install_dir*/SQLServer JTA/64-bit
64-bit AMD64 and Intel EM64T	*install_dir*/SQLServer JTA/x64-bit

where:

install_dir is your product installation directory.

SQL_Server_Root is your Microsoft SQL Server installation directory.

3. Start the Microsoft SQL Server instance.
4. From the database server, use the ISQL utility to run the .sql script. As a precaution, have your system administrator back up the master database before running the script.

   At a command prompt, run the script. For example:

   ```
   ISQL -Usa -Psa_password -Sserver_name -ilocation\instjdbc.sql
   ```

   where:

   sa_password is the password of the system administrator.

   server_name is the name of the server on which the Microsoft SQL Server database resides.

   location is the full path to instjdbc.sql. This script is located in the *install_dir*/SQLServer JTA directory, where *install_dir* is your product installation directory.

5. The script generates many messages. In general, these messages can be ignored; however, the system administrator should scan the output for any messages that may indicate an execution error. The last message should indicate that the script ran successfully. The script fails when there is insufficient space available in the master database to store the JDBC XA procedures or to log changes to existing procedures.

See also

Distributed transaction cleanup on page 90

Distributed transaction cleanup

Connections associated with distributed transactions can become orphaned if the connection to the server is lost before the transaction has completed. When connections associated with distributed transactions are orphaned, any locks held by the database for that transaction are maintained, which can cause data to become unavailable. By cleaning up distributed transactions, connections associated with those transactions are freed and any locks held by the database are released.

You can use the XAResource.recover() method to clean up distributed transactions that have been prepared, but not committed or rolled back. Calling this method returns a list of active distributed transactions that have been prepared, but not committed or rolled back. An application can use the list returned by the XAResource.recover method to clean up those transactions by explicitly committing them or rolling them back. The list of transactions returned by the XAResource.recover method does not include transactions that are active and have not been prepared.

In addition, the SQL Server driver supports the following methods of distributed transaction cleanup.

- Transaction timeout sets a timeout value that is used to audit active transactions. Any active transactions that have a life span greater than the specified timeout value are rolled back. Setting a transaction timeout allows distributed transactions to be cleaned up automatically based on the timeout value.
- Explicit transaction cleanup allows you to explicitly roll back any transactions left in an unprepared state based on a transaction group identifier. Explicit transaction cleanup provides more control than transaction timeout over when distributed transactions are cleaned up.

See also

JTA support: installing stored procedures on page 89

Transaction timeout

To set a timeout value for transaction cleanup, you use the XAResource.setTransactionTimeout method. Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

When a timeout value is set for transaction cleanup using the XAResource.setTransactionTimeout method, sqljdbc.dll periodically audits the list of active transactions for expired transactions. Any active transactions that have a life span greater than the timeout value are rolled back. If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

Setting the transaction timeout value too low means running the risk of rolling back a transaction that otherwise would have completed successfully. As a general guideline, set the timeout value to allow sufficient time for a transaction to complete under heavy traffic load.

Setting a value of `0` (default) disables transaction timeout cleanup.

Explicit transaction cleanup

The SQL Server driver allows you to associate an identifier with a group of transactions using the XATransactionGroup connection property. When you specify a transaction group ID, all distributed transactions initiated by the connection are identified by this ID.

Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

You can use the XAResource.recover method to roll back any transactions left in an unprepared state that match the transaction group ID on the connection used to call XAResource.recover. For example, if you specified XATransactionGroup=ACCT200 and called the XAResource.recover method on the same connection, any transactions left in an unprepared state with a transaction group ID of ACCT200 would be rolled back.

If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

When using explicit transaction cleanup, distributed transactions associated with orphaned connections, and the locks held by those connections, will persist until the application explicitly invokes them. As a general rule, applications should clean up orphaned connections at startup and when the application is notified that a connection to the server was lost.

Large object (LOB) support

Although Microsoft SQL Server does not define a Blob or Clob data type, the SQL Server driver allows you to return and update long data, specifically LONGVARBINARY and LONGVARCHAR data, using JDBC methods designed for Blobs and Clobs. When using these methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the Blob or Clob object.

Retrieving and updating long data using JDBC methods designed for Blobs and Clobs provides some of the same advantages as retrieving and updating Blobs and Clobs. For example, using Blobs and Clobs:

- Provides random access to data

- Allows searching for patterns in the data, such as returning long data that begins with a specific character string

To provide these advantages of Blobs and Clobs, data must be cached. Because data is cached, you will incur a performance penalty, particularly if the data is read once sequentially. This performance penalty can be severe if the size of the long data is larger than available memory.

Batch Inserts and Updates

The SQL Server driver implementation for batch Inserts and Updates is JDBC 3.0 compliant. When the SQL Server driver detects an error in a statement or parameter set in a batch Insert or Update, it generates a BatchUpdateException and continues to execute the remaining statements or parameter sets in the batch. The array of update counts contained in the BatchUpdateException contain one entry for each statement or parameter set. Any entries for statements or parameter sets that failed contain the value Statement.EXECUTE_FAILED.

Rowset support

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

Note: Auto-generated keys are not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

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 more information about using prepared statement pooling to optimize performance.

Null values

When the driver establishes a connection, the driver sets the Microsoft SQL Server database option ANSI_NULLS to on. This action ensures that the driver is compliant with the ANSI SQL standard, which makes developing cross-database applications easier.

By default, Microsoft SQL Server does not evaluate null values in SQL equality (=) or inequality (<>) comparisons or aggregate functions in an ANSI SQL-compliant manner. For example, the ANSI SQL specification defines that `col1=null` as shown in the following Select statement always evaluates to `false`:

```
SELECT * FROM table WHERE col1 = NULL
```

Using the default database setting (ANSI_NULLS=off, the same comparison evaluates to true instead of `false`.

Setting ANSI_NULLS to on changes how the database handles null values and forces the use of `IS NULL` instead of `=NULL`. For example, if the value of col1 in the following Select statement is null, the comparison evaluates to true:

```
SELECT * FROM table WHERE col1 IS NULL
```

In your application, you can restore the default Microsoft SQL Server behavior for a connection in the following ways:

Note: Setting ANSI_NULLS to off is not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

- Use the InitializationString property to specify the SQL command `set ANSI_NULLS off`. For example, the following URL ensures that the handling of null values is restored to the Microsoft SQL Server default for the current connection:

```
jdbc:datadirect:sqlserver://server1:1433;InitializationString=
set ANSI_NULLS off;DatabaseName=test
```

- Explicitly execute the following statement after the connection is established:

```
SET ANSI_NULLS OFF
```

DataDirect Bulk Load

In addition to supporting the native bulk protocols in SQL Server databases, the driver supports DataDirect Bulk Load. DataDirect Bulk Load allows you to perform bulk load operations by creating a `DDBulkLoad` object and using the methods provided by the `DDBulkLoad` interface in the `com.ddtek.jdbc.extensions` package for bulk load. You may want to use this method if you are developing a new application that needs to perform bulk load operations.

Important: Because a bulk load operation may bypass data integrity checks, your application must ensure that the data it is transferring does not violate integrity constraints in the database. For example, suppose you are bulk loading data into a database table and some of that data duplicates data stored as a primary key, which must be unique. The driver will not throw an exception to alert you to the error; your application must provide its own data integrity checks.

See also

Connection property descriptions on page 103

Using a DDBulkLoad object

The first step in performing a bulk load operation using a `DDBulkLoad` object is to create a `DDBulkLoad` object. A `DDBulkLoad` must be created with the `getInstance` method using the `DDBulkLoadFactory` class as shown in the following example.

```
import com.ddtek.jdbc.extensions.*
// Get SQLServer Connection
Connection con = DriverManager.getConnection(
   "jdbc:datadirect:sqlserver://MyServer:1433;User=User123;
    Password=secret;DatabaseName=MyDB");

// Get a DDBulkLoad object
DDBulkLoad bulkLoad = DDBulkLoadFactory.getInstance(con);
```

Once the `DDBulkLoad` object has been created, `DDBulkLoad` methods can be used to instruct the driver to obtain data from one location and load it to another location. The driver can obtain data either from a `ResultSet` object or from a CSV file.

Migrating data using a ResultSet object

The following steps would need to be taken to migrate data from Oracle to SQL Server using a `ResultSet` object.

Important: This scenario assumes that you are using the DataDirect Oracle driver to query the Oracle database and the DataDirect SQL Server driver to insert data to the SQL Server database.

---

1. The application creates a `DDBulkLoad` object.
2. The application executes a standard query on the Oracle database, and the Oracle driver retrieves the results as a `ResultSet` object.
3. The application instructs the SQL Server driver to load the data from the `ResultSet` object into SQL Server. (See "Loading data from a ResultSet object".)

Migrating data using a CSV file

The following steps would need to be taken to migrate data from Oracle to SQL Server using a CSV file.

1. The application creates a `DDBulkLoad` object.
2. The application instructs the Oracle driver to export the data from the Oracle database into a CSV file. (See "Exporting data to a CSV file.")
3. The application instructs the SQL Server driver to load data from the CSV file into SQL Server. (See "Loading data from a CSV file.")

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

See also

CSV files on page 98

Exporting data to a CSV file

Using a `BulkLoad` object, data can be exported either as a table or `ResultSet` object.

Exporting Data as a Table

To export data as a table, the application must specify the table name with the `setTableName` method and then export the data to a CSV file using the `export` method. For example, the following code snippet specifies the `GBMAXTABLE` table and exports it to a CSV file called `tmp.csv`.

```
bulkLoad.setTableName("GBMAXTABLE");
bulkLoad.export("tmp.csv");
```

Alternatively, you can create a file reference to the CSV file and use the `export` method to specify the file reference. For example:

```
File csvFile = new File ("tmp.csv");
bulkLoad.export(csvFile);
```

Exporting Data as a ResultSet object

To export data as a `ResultSet` object, the application must first create a file reference to a CSV file, and then, using the `export` method, specify the `ResultSet` object and the file reference. For example, the following code snippet creates the `tmp.csv` file reference and then specifies `MyResults` (the `ResultSet` object be exported).

```
File csvFile = new File ("tmp.csv");
bulkLoad.export(MyResults, csvFile);
```

If the CSV file does not already exist, the driver creates it when the `export` method is executed. The driver also creates a bulk load configuration file, which describes the structure of the CSV file.

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

See also

CSV files on page 98

Loading data from a ResultSet object

The `setTableName` method should be used to specify the table into which you want to load the data. Then, the `load` method is used to specify the `ResultSet` object that contains the data you are loading. For example, the following code snippet loads data from a `ResultSet` object named `MyResults` into a table named `GBMAXTABLE`

```
bulkLoad.setTableName("GBMAXTABLE");
bulkLoad.load(MyResults);
```

This example loads the first column in the result set to the `ColName1` column, the second column in the result set to the `ColName2` column, and so on.

You can use the BulkLoadBatchSize property to specify the number of rows the driver loads 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.

See also

BulkLoadBatchSize on page 127

Loading data from a CSV file

The `setTableName` method should be used to specify the table into which you want to load the data. Then, the `load` method is used to specify the CSV file that contains the data you are loading. For example:

```
bulkLoad.setTableName("GBMAXTABLE");
bulkLoad.load("tmp.csv");
```

Alternatively, you can create a file reference to the CSV file, and use the load() method to specify the file reference:

```
File csvFile = new File("tmp.csv");
bulkLoad.load(csvFile);
```

For the Salesforce driver, you also can specify the columns to which you want to load the data. This example loads the first column in the CSV file to the `ColName1` column, the second column in the CSV file to the `ColName2` column, and the third column in the CSV file to the `ColName3` column:

```
bulkLoad.setTableName("GBMAXTABLE(ColName1, ColName2, ColName3)");
bulkLoad.load("tmp.csv");
```

Use the BulkLoadBatchSize property to specify the number of rows the driver loads 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.

See also

CSV files on page 98
BulkLoadBatchSize on page 127

Specifying the bulk load operation

You can specify which type of bulk load operation will be performed when a load method is called by setting the operation property using the `setProperties` method of the `DDBulkLoad` interface. The operation property accepts the following values: `insert`, `update`, `delete` and `upsert`. The default value is `insert`.

The following example changes the type of bulk load operation to update.

```
DDBulkLoad bulkLoad =
com.ddtek.jdbc.extensions.DDBulkLoadFactory.getInstance(connection);
Properties props = new Properties();
props.put("operation", "update");
bulkLoad.setProperties(props);
```

Logging

If logging is enabled for bulk load, a log file records information for each bulk load operation. Logging is enabled by specifying a file name and location for the log file using the `setLogFile` method.

The log file records the following types of information about each bulk load operation.

- Total number of rows that were read
- Total number of rows that successfully loaded

  Note: The total number of rows that successfully loaded is not provided for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

- Total number of rows that failed to load

For example, the following log file shows that the 11 rows read were all successfully loaded.

```
/*----- Load Started: <Feb 25, 2009 11:20:09 AM EST>---------------------*/
Total number of rows read 11
Total number of rows successfully loaded 11
Total number of rows that failed to load 0
```

Enabling Logging on Windows

To enable logging using a log file named `bulk_load.log` located in the `C:\temp` directory, you would specify:

```
bulkLoad.setLogFile(C:\\temp\\bulk_load.log)
```

Note: 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: `C:\\temp\\bulk_load.log`.

Enabling Logging on UNIX/Linux

To enable logging using a log file named `bulk_load.log` located in the `/tmp` directory, you would specify:

```
bulkLoad.setLogFile(/tmp/bulk_load.log)
```

CSV files

As described in "Exporting data to a CSV file," the driver can create a CSV file by exporting data from a table or `ResultSet` object. For example, suppose you want to export data from a 4-column table named `GBMAXTABLE` into a CSV file. The contents of the CSV file, named `GBMAXTABLE.csv`, might look like the following example:

```
1,0x6263,"bc","bc"
2,0x636465,"cde","cde"
3,0x64656667,"defg","defg"
4,0x6566676869,"efghi","efghi"
5,0x666768696a6b,"fghijk","fghijk"
6,0x6768696a6b6c6d,"ghijklm","ghijklm"
7,0x68696a6b6c6d6e6f,"hijklmno","hijklmno"
8,0x696a6b6c6d6e6f7071,"ijklmnopq","ijklmnopq"
9,0x6a6b6c6d6e6f70717273,"jklmnopqrs","jklmnopqrs"
10,0x6b,"k","k"
```

See also

Exporting data to a CSV file on page 95

Bulk load configuration file

Each time data is exported to a CSV file, a bulk load configuration file is created. This file has the same name as the CSV file, but with an `.xml` extension (for example, `GBMAXTABLE.xml`).

In its metadata, the bulk load configuration file defines the names and data types of the columns in the CSV file based on those in the table or `ResultSet` object. It also defines other data properties, such as the source code page for character data types, precision and scale for numeric data types, and nullability for all data types. The format of `GBMAXTABLE.xml` might look like the following example.

Note: If the driver cannot read a bulk load configuration file (for example, because it was inadvertently deleted), the driver reads all data as character data. The character set used by the driver is UTF-8.

```
<?xml version="1.0" encoding="utf-8"?>
<table codepage="UTF-8" xsi:noNamespaceSchemaLocation=
"http://media.datadirect.com/download/docs/ns/bulk/BulkData.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
   <column datatype="DECIMAL" precision="38" scale="0" nullable="false">
      INTEGERCOL</column>
   <column datatype="VARBINARY" length="10"
nullable="true">VARBINCOL</column>
   <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
      externalfilecodepage="Windows-1252"
nullable="true">VCHARCOL</column>
   <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
      externalfilecodepage="Windows-1252"
nullable="true">UNIVCHARCOL</column>
</row>
</table>
```

Bulk load configuration file schema

The bulk load configuration file must conform to the bulk load configuration XML schema defined at the following Web site.

http://media.datadirect.com/download/docs/ns/bulk/BulkData.xsd

The driver throws an exception if either of the following circumstances occur.

- If the driver performs a data export and the CSV file cannot be created
- If the driver performs a bulk load operation and the driver detects that the CSV file does not comply with the XML schema described in the bulk load configuration file

Character set conversions

When you export data from a database to a CSV file, the CSV file uses the same code page as the table from which the data was exported. If the CSV file and the target table use different code pages, performance for bulk load operations can suffer because the driver must perform a character set conversion.

To avoid character set conversions, your application can specify which code page to use for the CSV file when exporting data. For example, if the source database table uses a SHIFT-JIS code page and the target table uses a EUC-JP code page, specify `setCodePage("EUC_JP")` to ensure that the CSV file will use the same code page as the target table.

You can specify any of the following code pages.

Note: If the code page you need to use is not listed, contact Customer Support to request support for that code page.

US_ASCII	IBM273	IBM01140
ISO_8859_1	IBM277	IBM01141
ISO_8859_2	IBM278	IBM01142
ISO_8859_3	IBM280	IBM01143
ISO_8859_4	IBM284	IBM01144
ISO_8859_5	IBM285	IBM01145
ISO_8859_6	IBM290	IBM01146
ISO_8859_7	IBM297	IBM01147
ISO_8859_8	IBM420	IBM01148
ISO_8859_9	IBM424	IBM01149
JIS_Encoding	IBM500	WINDOWS-1250
Shift_JIS	IBM851	WINDOWS-1251
EUC_JP	IBM855	WINDOWS-1252
KS_C_5601	IBM857	WINDOWS-1253
ISO_2022_KR	IBM860	WINDOWS-1254
EUC_KR	IBM861	WINDOWS-1255
ISO_2022_JP	IBM863	WINDOWS-1256
GB2312	IBM864	WINDOWS-1257
ISO_8859_13	IBM865	WINDOWS-1258
ISO_8859_15	IBM869	WINDOWS-854
GBK	IBM870	IBM-939
IBM850	IBM871	IBM-943_P14A-2000
IBM852	IBM1026	IBM-4396
IBM437	KOI8_R	IBM-5026
IBM862	HZ_GB_2312	IBM-5035
Big5	IBM866	UTF-8
MACINTOSH	IBM775	UTF-16LE
IBM037	IBM00858	UTF-16BE

External overflow files

When you export data into a CSV file, you can choose to create one large file or multiple smaller files. For example, if you are exporting BLOB data that is a total of several GB, you may want to break the data that into multiple smaller files of 100 MB each.

If the values set by the `setCharacterThreshold` or `setBinaryThreshold` methods are exceeded, separate files are generated to store character or binary data, respectively. Overflow files are located in the same directory as the CSV file.

The format for overflow file names is:

`CSV_file_name``.xxxxxx.lob`

where:

`CSV_file_name` is the name of the CSV file.

`xxxxxx` is a 6-digit number that increments an overflow file.

For example, if multiple overflow files are created for a CSV file named `CSV1`, the file names would look like this:

```
CSV1.000001.lob
CSV1.000002.lob
CSV1.000003.lob
...
```

If the overflow file contains character data, the code page used by the file is the code page specified in the bulk load configuration file for the CSV file.

Discard file

If the driver was unable to load rows into the database for a bulk load operation from a CSV file, it can record all the rows that were not loaded in a discard file. The contents of the discard file is in the same format as that of the CSV file. After fixing reported issues in the discard file, the bulk load can be reissued, using the discard file as the CSV file.

A discard file is created by specifying a file name and location for the discard file using the `setDiscardFile` method.

Creating a discard file on Windows

To create a discard file named `discard.csv` located in the `C:\temp` directory, you would specify:

```
bulkLoad.setDiscardFile(C:\\temp\\discard.csv)
```

Note: 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: `C:\\temp\\discard.csv`.

Creating a discard file on UNIX/Linux

To create a discard file named `discard.csv` located in the `/tmp` directory, you would specify:

```
bulkLoad.setDiscardFile(/tmp/discard.csv)
```

Inserts into IDENTITY columns for data replication

The driver supports inserts into IDENTITY columns for data replication. This functionality might be useful when publishing data from an earlier version of SQL Server. By default, the columns defined as IDENTITY contain auto-generated numbers for new rows inserted. Identity columns might be included as a part of the primary key, therefore, it is important to avoid duplicate values in the these columns. By specifying a value, you can maintain the row identity of the source column in the destination column in SQL Server.

To enable inserts into IDENTITY columns, set the EnableReplicationUser connection property to `true` (Enabled). When the property is set to `true` (Enabled), explicit values can be inserted into IDENTITY columns defined as NOT FOR REPLICATION. For inserts to succeed, IDENTITY columns must defined as NOT FOR REPLICATION.

See also

EnableReplicationUser on page 143

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.

Required properties

The following table summarizes properties required for connecting.

Property	Data Source Method	Default
PortNumber on page 164	`getPortNumber()` `setPortNumber(Integer)`	`1433`
ServerName on page 171	`getServerName()` `setServerName(String)`	None

Authentication properties

The following table summarizes properties required for authentication.

Property	Data Source Method	Default
AccessToken on page 114	`getAccessToken()` `setAccessToken(String)`	None
ActiveDirectoryPrincipalID on page 115	`getActiveDirectoryPrincipalID()` `setActiveDirectoryPrincipalID(String)`	Empty string
ActiveDirectoryPrincipalSecret on page 116	`getActiveDirectoryPrincipalSecret()` `setActiveDirectoryPrincipalSecret(String)`	Empty string
AuthenticationMethod on page 125	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`auto`
Domain on page 141	`getDomain()` `setDomain(String)`	None
GSSCredential on page 150	`getGSSCredential()` `setGSSCredential(String)`	Null

Property	Data Source Method	Default
LoginConfigName on page 156	`getLoginConfigName()` `setLoginConfigName(String)`	`JDBC_DRIVER_01`
Password on page 163	`getPassword()` `setPassword(String)`	None
ServicePrincipalName on page 172	`getServicePrincipalName()` `setServicePrincipalName(String)`	Driver builds value based on environment
User on page 182	`getUser()` `setUser(String)`	None

Data encryption properties

The following table summarizes properties required for encrypting data.

Property	Data Source Method	Default
CryptoProtocolVersion on page 135	`getCryptoProtocolVersion()` `setCryptoProtocolVersion(String)`	None
EncryptionMethod on page 144	`getEncryptionMethod()` `setEncryptionMethod(String)`	`noEncryption`
HostNameInCertificate on page 151	`getHostNameInCertificate()` `setHostNameInCertificate(String)`	Empty string
TrustStore on page 181	`getTrustStore()` `setTrustStore(String)`	None
TrustStorePassword on page 181	`getTrustStorePassword()` `setTrustStorePassword(String)`	None
ValidateServerCertificate on page 184	`getValidateServerCertificate()` `setValidateServerCertificate(Boolean)`	`true`

Failover properties

The following table summarizes properties which can be used to implement failover.

Property	Data Source Method	Default
AlternateServers on page 122	`getAlternateServers()` `setAlternateServers(String)`	None
ConnectionRetryCount on page 133	`getConnectionRetryCount()` `setConnectionRetryCount(Integer)`	5
ConnectionRetryDelay on page 134	`getConnectionRetryDelay()` `setConnectionRetryDelay(Integer)`	1 (second)
FailoverGranularity on page 145	`getFailoverGranularity()` `setFailoverGranularity(String)`	`nonAtomic`
FailoverMode on page 146	`getFailoverMode()` `setFailoverMode(String)`	`connect`
FailoverPreconnect on page 147	`getFailoverPreconnect()` `setFailoverPreconnect(Boolean)`	`false`
LoadBalancing on page 155	`getLoadBalancing()` `setLoadBalancing(Boolean)`	`false`
MultiSubnetFailover on page 160	`getMultiSubnetFailover()` `setMultiSubnetFailover(Boolean)`	`false`

Bulk load properties

The following table summarizes properties used to configure bulk operations.

Property	Data Source Method	Default
BulkLoadBatchSize on page 127	`getBulkLoadBatchSize()` `setBulkLoadBatchSize(Long)`	`1000` (rows)
BulkLoadOptions on page 128	`getBulkLoadOptions()` `setBulkLoadOptions(Long)`	2
EnableBulkLoad on page 142	`getEnableBulkLoad()` `setEnableBulkLoad(Boolean)`	`false`

Proxy server properties

The following table summarizes properties used for proxy server connections.

Property	Data Source Method	Default
ProxyHost on page 165	`getProxyHost()` `setProxyHost(String)`	None
ProxyPassword on page 166	`getProxyPassword()` `setProxyPassword(String)`	None
ProxyPort on page 167	`getProxyPort()` `setProxyPort(Integer)`	`0` which means that the default value is determined by whether the value specified for the ProxyHost property is an HTTP or HTTPS URL. For HTTP: `80` For HTTPS: `443`
ProxyUser on page 167	`getProxyUser()` `setProxyUser(String)`	None

Data type properties

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

Property	Data Source Method	Default
ConvertNull on page 135	`getConvertNull()` `setConvertNull(Integer)`	`1` (data type check is performed if column value is null)
DateTimeInputParameterType on page 137	`getDateTimeInputParameterType()` `setDateTimeInputParameterType(String)`	`auto`
DateTimeOutputParameterType on page 138	`getDateTimeOutputParameterType()` `setDateTimeOutputParameterType(String)`	`auto`
DescribeInputParameters on page 139	`getDescribeInputParameters()` `setDescribeInputParameters(String)`	`noDescribe`
DescribeOutputParameters on page 140	`getDescribeOutputParameters()` `setDescribeOutputParameters(String)`	`noDescribe`
FetchTSWTZAsTimestamp on page 148	`getFetchTSWTZAsTimestamp()` `setFetchTSWTZAsTimestamp(Boolean)`	`false`
FetchTWFSasTime on page 149	`getFetchTWFSasTime()` `setFetchTWFSasTime(Boolean)`	`false`

Property	Data Source Method	Default
JavaDoubleToString on page 154	`getDateTimeOutputParameterType()` `setJavaDoubleToString(Boolean)`	`false`
JDBCBehavior on page 155	`getJDBCBehavior()` `setJDBCBehavior(Integer)`	`1`
XMLDescribeType on page 186	`getXMLDescribeType()` `setXMLDescribeType(String)`	None

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
EnableCancelTimeout on page 142	`getEnableCancelTimeout()` `setEnableCancelTimeout(Boolean)`	`false`
LoginTimeout on page 157	`getLoginTimeout()` `setLoginTimeout(Integer)`	`0`
QueryTimeout on page 168	`getQueryTimeout()` `setQueryTimeout(Integer)`	`0`

Statement pooling properties

The following table summarizes the statement pooling connection properties.

Property	Data Source Method	Default
ImportStatementPool on page 152	`getImportStatementPool()` `setImportStatementPool(String)`	None
MaxPooledStatements on page 159	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 169	`getRegisterStatementPoolMonitorMBean()` `setRegisterStatementPoolMonitorMBean(Boolean)`	`false`

Client information properties

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

Property	Data Source Method	Default
AccountingInfo on page 114	`getAccountingInfo()` `setAccountingInfo(String)`	None
ApplicationName on page 124	`getApplicationName()` `setApplicationName(String)`	None
ClientHostName on page 129	`getClientHostName()` `setClientHostName(String)`	None
ClientUser on page 130	`getClientUser()` `setClientUser(String)`	None
NetAddress on page 161	`getNetAddress()` `setNetAddress(String)`	`000000000000`
ProgramID on page 164	`getProgramID()` `setProgramID(String)`	None

Always encrypted properties

The following table summarizes the connection properties related to Always Encrypted functionality.

Property	Data Source Method	Default
AEKeyCacheTTL on page 117	`getAEKeyCacheTTL()` `setAEKeyCacheTTL(Long)`	`7200`
ColumnEncryption on page 132	`getColumnEncryption()` `setColumnEncryption(String)`	`Disabled`
AEKeystoreClientSecret on page 118	`getAEKeystoreClientSecret()` `setAEKeystoreClientSecret(String)`	None
AEKeystorePrincipalId on page 120	`getAEKeystorePrincipalId()` `setAEKeystorePrincipalId(String)`	None
AEKeystoreLocation on page 119	`getAEKeystoreLocation()` `setAEKeystoreLocation(String)`	None
AEKeystoreSecret on page 121	`getAEKeystoreSecret()` `setAEKeystoreSecret(String)`	None

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
AlwaysReportTriggerResults on page 123	`getAlwaysReportTriggerResults()` `setAlwaysReportTriggerResults(Boolean)`	`false`
ApplicationIntent on page 123	`getApplicationIntent()` `setApplicationIntent(String)`	`ReadWrite`
CatalogOptions on page 129	`getCatalogOptions()` `setCatalogOptions(Integer)`	`0`
CodePageOverride on page 131	`getCodePageOverride()` `setCodePageOverride(String)`	None
DatabaseName on page 137	`getDatabaseName()` `setDatabaseName(String)`	None
EnableReplicationUser on page 143	`getEnableReplicationUser()` `setEnableReplicationUser(Boolean)`	`false` (Disabled)
InitializationString on page 152	`getInitializationString()` `setInitializationString(String)`	None
InsensitiveResultSetBufferSize on page 153	`getInsensitiveResultSetBufferSize()` `setInsensitiveResultSetBufferSize(Integer)`	`2048` KB
LongDataCacheSize on page 158	`getLongDataCacheSize()` `setLongDataCacheSize(Integer)`	`2048`
PacketSize on page 162	`getPacketSize()` `setPacketSize(Integer)`	`-1`
ResultSetMetaDataOptions on page 170	`getResultSetMetaDataOptions()` `setResultSetMetaDataOptions(Integer)`	`0`
SelectMethod on page 170	`getSelectMethod()` `setSelectMethod(String)`	`direct`
SnapshotSerializable on page 174	`getSnapshotSerializable()` `setSnapshotSerializable(Boolean)`	`false`

Property	Data Source Method	Default
SpyAttributes on page 175	`getSpyAttributes()` `setSpyAttributes(String)`	None
SuppressConnectionWarnings on page 179	`getSuppressConnectionWarnings()` `setSuppressConnectionWarnings(Boolean)`	`false`
TransactionMode on page 179	`getTransactionMode()` `setTransactionMode(String)`	`implicit`
TruncateFractionalSeconds on page 180	`getTruncateFractionalSeconds()` `setTruncateFractionalSeconds(Boolean)`	`true`
UseServerSideUpdatableCursors on page 183	`getUseServerSideUpdatableCursors()` `setUseServerSideUpdatableCursors(Boolean)`	`false`
XATransactionGroup on page 185	`getXATransactionGroup()` `setXATransactionGroup(String)`	None

For details, see the following topics:

- AccessToken
- AccountingInfo
- ActiveDirectoryPrincipalID
- ActiveDirectoryPrincipalSecret
- AEKeyCacheTTL
- AEKeystoreClientSecret
- AEKeystoreLocation
- AEKeystorePrincipalId
- AEKeystoreSecret
- AlternateServers
- AlwaysReportTriggerResults
- ApplicationIntent
- ApplicationName
- AuthenticationMethod
- BulkLoadBatchSize
- BulkLoadOptions

- CatalogOptions
- ClientHostName
- ClientUser
- CodePageOverride
- ColumnEncryption
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CryptoProtocolVersion
- DatabaseName
- DateTimeInputParameterType
- DateTimeOutputParameterType
- DescribeInputParameters
- DescribeOutputParameters
- Domain
- EnableBulkLoad
- EnableCancelTimeout
- EnableReplicationUser
- EncryptionMethod
- FailoverGranularity
- FailoverMode
- FailoverPreconnect
- FetchTSWTZAsTimestamp
- FetchTWFSasTime
- GSSCredential
- HostNameInCertificate
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- JDBCBehavior
- LoadBalancing
- LoginConfigName

- LoginTimeout
- LongDataCacheSize
- MaxPooledStatements
- MultiSubnetFailover
- NetAddress
- PacketSize
- Password
- PortNumber
- ProgramID
- ProxyHost
- ProxyPassword
- ProxyPort
- ProxyUser
- QueryTimeout
- RegisterStatementPoolMonitorMBean
- ResultSetMetaDataOptions
- SelectMethod
- ServerName
- ServicePrincipalName
- SnapshotSerializable
- SpyAttributes
- StringInputParameterType
- StringOutputParameterType
- SuppressConnectionWarnings
- TransactionMode
- TruncateFractionalSeconds
- TrustStore
- TrustStorePassword
- User
- UseServerSideUpdatableCursors
- ValidateServerCertificate
- XATransactionGroup
- XMLDescribeType

AccessToken

Purpose

Specifies the access token required to authenticate to a SQL Server instance for access token based Entra ID authentication. It can only be passed as a `Properties` or `DataSource` object. If an access token value is specified, the AuthenticationMethod property is automatically set to the default (`AuthenticationMethod=auto`). Refer to SQL Server documentation to know how to obtain an access token.

Valid Values

`string`

where:

`string`

is the access token you have obtained from Entra ID.

Notes

- If a value for the AccessToken property is specified, the driver will ignore any other authentication method to establish a connection.
- If accessToken is specified, driver will use encryptionMethod=SSL even if encryptionMethod is not specified or specified with "NoEncryption" option.
- Setting accessToken is recommended using a `Properties` or `DataSource` only.

Data Source Method

```
public String getAccessToken()
```

```
public void setAccessToken(String)
```

Default

None

Data Type

String

AccountingInfo

Purpose

Defines accounting information. This value is stored locally and is used for database administration/monitoring purposes.

Valid values

string

where:

string

is the accounting information.

Data source method

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

Default

Empty string

Data type

String

See also

-

- **Note:** For more information on connection client information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference*.

- Connection property descriptions on page 103

ActiveDirectoryPrincipalID

Purpose

Specifies the Microsoft Entra ID Identity (PrincipalID) of the Azure SQL logical server. This property is used when authenticating the service principal user with Entra ID (`AuthenticationMethod=ActiveDirectoryServicePrincipal`). The value of this property is the same as the PrincipalID in the Entra ID interface.

Valid values

string

where:

string

is PrincipalID used for service principal user authentication with Entra ID.

Notes

- To retrieve the access token used for authentication, both the principal ID (ActiveDirectoryPrincipalID) and client secret (ActiveDirectoryPrincipalSecret) must be specified.

Data source method

```
public String getActiveDirectoryPrincipalID()
```

```
public void setActiveDirectoryPrincipalID(String)
```

Default

Empty string

Data type

String

See also

- ActiveDirectoryPrincipalSecret on page 116
- Microsoft Entra ID authentication on page 51

ActiveDirectoryPrincipalSecret

Purpose

Specifies the client secret for your Microsoft Entra ID (Azure Active Directory) application. This property is used when authenticating the service principal user with Entra ID (`AuthenticationMethod=ActiveDirectoryServicePrincipal`). The value of this property is the same as the ClientSecret in the Entra ID interface.

Valid values

`string`

where:

`string`

is the client secret used for service principal authentication with Entra ID.

Notes

- To retrieve the access token used for authentication, both the principal ID (ActiveDirectoryPrincipalID) and client secret (ActiveDirectoryPrincipalSecret) must be specified.

Data source method

```
public String getActiveDirectoryPrincipalSecret()
```

```
public void setActiveDirectoryPrincipalSecret(String)
```

Default

Empty string

Data type

String

See also

- ActiveDirectoryPrincipalID on page 115
- Microsoft Entra ID authentication on page 51

AEKeyCacheTTL

Purpose

Specifies the length of time, in seconds, column encryption keys live in the cache before the driver deletes them. This option is used when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`).

Valid Values

`0` | *x*

where:

x

is the number of seconds the driver stores a column encryption key in the cache.

Behavior

If set to `-1`, the driver caches column encryption keys for the life of the connection. The keys are deleted when the connection is closed or added to the connection pool.

If set to `0`, the driver does not cache column encryption keys.

If set to *x*, the driver caches column encryption keys for the specified number of seconds before deleting them. The timer starts for a key when it is first accessed and added to the cache. The timer does not reset if you access it after it has been added to the cache. The keys are deleted when the timer expires, or the connection is closed or added to the connection pool.

Notes

- Column encryption keys do not persist beyond the life of the connection. When a connection is closed, the driver purges the cache, leaving no column encryption key data in memory.
- Caching column encryption keys can provide performance gains by reducing the overhead associated with fetching and decrypting the keys for the same data multiple times during a connection. Specifying larger values for this property increases the length of time that a column encryption key persists in the cache; therefore, improving performance in some scenarios. Note that column encryption keys are designed to be deleted from the cache as a security measure and should not be configured to live for long periods of time.

Data source method

```
public Long getAEKeyCacheTTL()
public void setAEKeyCacheTTL(Long)
```

Default

```
7200
```

Data Type

Long

See Also

- ColumnEncryption on page 132
- Always Encrypted on page 64
- Connection property descriptions on page 103
- Performance considerations on page 77

AEKeystoreClientSecret

Purpose

Specifies the Client Secret used to authenticate against the Azure Key Vault. This property is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and Azure Key Vault is the keystore provider. The Azure Key Vault stores the column master key used for Always Encrypted functionality. To access the column master key from the Azure Key Vault, the Client Secret and principal ID must be provided.

Valid Values

`client_secret`

where:

`client_secret`

is the Client Secret used to authenticate against the Azure Key Vault.

Notes

- To specify the principal ID, use the AEKeystorePrincipalId connection property.

Data source method

```
public String getAEKeystoreClientSecret()
public void setAEKeystoreClientSecret(String)
```

Default

None

Data Type

String

See Also

- ColumnEncryption on page 132
- AEKeystorePrincipalId on page 120
- Always Encrypted on page 64

AEKeystoreLocation

Purpose

Specifies the absolute path to the Java KeyStore file. This property is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and the Java KeyStore is the keystore provider. The Java KeyStore file contains the column master key used for Always Encrypted functionality.

Valid Values

`java_keystore_path`

where:

`java_keystore_path`

is the absolute path to the Java KeyStore file.

Notes

- To specify the password for the Java KeyStore file, use the AEKeystoreSecret connection property.
- This property is required when encryption keys are stored in a Java KeyStore and Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`).
- The value for this property is ignored when Always Encrypted is disabled (`ColumnEncryption=Disabled`).
- The driver supports the JKS and PKCS12 file formats.

Data source method

```
public String getAEKeystoreLocation()
```

```
public void setAEKeystoreLocation(String)
```

Default

None

Data Type

String

See Also

- ColumnEncryption on page 132

- AEKeystoreSecret on page 121
- Always Encrypted on page 64
- Connection property descriptions on page 103

AEKeystorePrincipalId

Purpose

Specifies the principal ID used to authenticate against the Azure Key Vault. This property is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and Azure Key Vault is the keystore provider. The Azure Key Vault stores the column master key used for Always Encrypted functionality. To access the column master key from the Azure Key Vault, the principal ID and Client Secret must be provided.

Valid Values

`principal_id`

where:

`principal_id`

is the Application ID created during Azure App Registration and used to authenticate against the Azure Key Vault.

Notes

- To specify the Client Secret, use the AEKeystoreClientSecret connection property.
- The driver currently supports only Azure App Registration as the principal ID.
- This property is used only when the Azure Key Vault is specified as the keystore provider by column metadata or in statement parameters.

Data source method

```
public String getAEKeystorePrincipalId()
public void setAEKeystorePrincipalId(String)
```

Default

None

Data Type

String

See Also

- ColumnEncryption on page 132
- AEKeystoreClientSecret on page 118
- Always Encrypted on page 64

AEKeystoreSecret

Purpose

Specifies the password used to access the Java KeyStore file. This property is used only when Always Encrypted is enabled (`ColumnEncryption=Enabled` | `ResultsetOnly`) and the Java KeyStore is the keystore provider. The Java KeyStore contains the column master key used for Always Encrypted functionality.

Valid Values

`keystore_password`

where:

`keystore_password`

is the password used to access the key(s) in the Java keystore.

Notes

- This property is used to access the key(s) in the Java KeyStore file specified by the AEKeystoreLocation connection property.
- If no value is specified for this property, an empty string is passed to the Java KeyStore file.
- The value for this property is ignored when Always Encrypted is disabled (`ColumnEncryption=Disabled`).

Data source method

```
public String getAEKeystoreSecret()
```

```
public void setAEKeystoreSecret(String)
```

Default

Empty string

Data Type

String

See Also

- ColumnEncryption on page 132
- AEKeystoreLocation on page 119
- Always Encrypted on page 64
- Connection property descriptions on page 103

AlternateServers

Purpose

A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See FailoverMode on page 146 for information about choosing a failover method.

Valid values

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

Behavior

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

The driver allows only one optional connection property, DatabaseName.

Notes

If using failover with Microsoft Cluster Server (MSCS), which determines the alternate server for failover instead of the driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server1:1433;DatabaseName=TEST)
```

Example

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

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

Data source method

```
public String getAlternateServers()
```

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

Default

None

Data type

String

See also

- FailoverMode on page 146

-

- **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*.

AlwaysReportTriggerResults

Purpose

Determines how the driver reports results that are generated by database triggers (procedures that are stored in the database and executed, or fired, when a table is modified).

Valid values

`true` | `false`

Behavior

If set to `true`, the driver returns all results, including results that are generated by triggers. Multiple trigger results are returned sequentially. Use the Statement.getMoreResults() method to return individual trigger results. Warnings and errors are reported in the results as they are encountered.

If set to `false`, the driver does not report trigger results if the statement is a single Insert, Update, Delete, Create, Alter, Drop, Grant, Revoke, or Deny statement. The only result that is returned is the update count that is generated by the statement that was executed (if errors do not occur). Although trigger results are ignored, any errors that are generated by the trigger are reported. Any warnings that are generated by the trigger are enqueued. If errors are reported, the update count is not reported.

Data source method

```
public Boolean getAlwaysReportTriggerResults()
```

```
public void setAlwaysReportTriggerResults(Boolean)
```

Default

`false`

Data type

Boolean

ApplicationIntent

Purpose

Specifies whether the driver connects to read-write databases or requests read-only routing to connect to read-only database replicas. Read-only routing only applies to connections in Microsoft SQL Server 2012 where AlwaysOn Availability Groups have been deployed.

Valid values

`ReadWrite | ReadOnly`

Behavior

If set to `ReadWrite`, the driver connects to a read-write node in the AlwaysOn environment.

If set to `ReadOnly`, the driver requests read-only routing and connects to the read-only database replicas specified by the server.

Notes

By setting applicationIntent to `ReadOnly` and querying read-only database replicas when possible, you shift load away from the read-write nodes of your database cluster to read-only nodes.

Data source method

```
public String getApplicationIntent()
```

```
public void setApplicationIntent(String)
```

Default

`ReadWrite`

Data type

String

See also

Always On Availability Groups on page 77

ApplicationName

Purpose

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

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 method

```
public String getApplicationName()
```

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

Default

Empty string

Data type

String

See also

- Connection property descriptions on page 103
-

- **Note:** For more information on client information associated with a connection, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference*.

AuthenticationMethod

Purpose

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

Valid values

`activeDirectoryPassword` | `ActiveDirectoryServicePrincipal` | `auto` | `kerberos` | `ntlm` | `ntlmjava` | `ntlm2java` | `userIdPassword` | `ActiveDirectoryManagedIdentity`

Behavior

If set to `ActiveDirectoryPassword`, the driver uses Entra ID user name and password authentication when establishing a connection to Azure. In addition to specifying a user ID and password, a value must be specified for the HostNameInCertificate property. All communications to the service are encrypted using SSL.

If set to `ActiveDirectoryServicePrincipal`, the driver uses Entra ID service principal user authentication when establishing a connection to Azure. This setting requires the ActiveDirectoryPrincipalID and ActiveDirectoryPrincipalSecret properties to be specified. All communications to the service are encrypted using SSL.

If set to `ActiveDirectoryManagedIdentity`, the driver uses managed identity authentication when establishing a connection and accessing Entra ID resources. The User property provides the client ID of the user assigned managed identity. If a user is not specified, the driver authenticates using a system-assigned managed identity.

If set to `auto`, the driver uses SQL Server authentication, access token authentication, or Kerberos authentication based on the following criteria.

- If a user ID and password is specified, the driver uses SQL Server authentication when establishing a connection. The User property provides the user ID. The Password property provides the password.

- If a user ID and password is not specified, the driver uses Kerberos authentication when establishing a connection.
- If an access token value is specified, then authentication using the access token takes precedence over other authentication methods. Refer to Access token authentication on page 49 for details.

If set to `kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the User and Password properties. The driver uses the authentication technology based on the value specified for the LoginConfigName property to establish a Kerberos connection.

If set to `ntlm`, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. User ID and password are optional. If user ID and password are specified, those credentials will be used. Otherwise, the current OS user credentials will be used. This value is supported for Windows clients only.

If set to `ntlmjava`, the driver uses NTLMv1 or NTLMv2 depending on the size of the NTLM password. NTLMv1 is used if the password is 14 bytes or less; NTLMv2 is used if the password is more than 14 bytes. A user ID and password must also be specified. If the user ID and password are unspecified, the driver throws an exception. In addition, the driver requires the name of the domain server that administers the database server. You can specify it using the Domain property. If the Domain property is unspecified, the driver attempts to determine the domain server name from the User property. If no domain is specified, the driver throws an exception.

If set to `ntlm2java`, the driver uses NTLMv2 authentication. A user ID and password must also be specified. If the user ID and password are unspecified, the driver throws an exception. In addition, the driver requires the name of the domain server that administers the database server. You can specify it using the Domain property. If the Domain property is unspecified, the driver attempts to determine the domain server name from the User property. If no domain is specified, the driver throws an exception.

If set to `userIdPassword`, the driver uses SQL Server authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. If a user ID is not specified, the driver throws an exception.

Notes

- If your are configuring your environment for Kerberos constrained delegation, AuthenticationMethod must be set to `kerberos`.
- The User property provides the user ID. The Password property provides the password.
- When using Entra ID authentication (`AuthenticationMethod=ActiveDirectoryPassword`), the driver requires root CA certificates to establish an SSL connection to a database. The driver determines the location of the truststore containing the required certificates by using the default JRE `cacerts` file, unless a different file has been specified by the `javax.net.ssl.trustStore` Java system property. The truststore location cannot be specified using the driver's Truststore property.
- If you specify `AuthenticationMethod=ntlmjava` when the LMCompatabilityLevel has been restricted to NTLMv2, an error will be returned. When the LMCompatabilityLevel has been restricted to NTLMv2, AuthenticationMethod must be set to `ntlm2java`.

Data source method

```
public String getAuthenticationMethod()
```

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

Default

`auto`

Data type

String

See also

Authentication on page 48
Connection property descriptions on page 103

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.

Notes

- This property suggests the number of rows regardless whether using a DDBulkLoad object or using native bulk protocols in the database for batch inserts.
- The DDBulkObject.setBatchSize() method overrides the value set by this property.

Data source method

```
public Long getBulkLoadBatchSize()
```

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

Default

`1000` (rows)

Data type

Long

See also

- DataDirect Bulk Load on page 94
- Connection property descriptions on page 103

BulkLoadOptions

Purpose

Enables bulk load protocol options for batch inserts that the driver can take advantage of when EnableBulkLoad is set to a value of `true`.

Valid values

This value is the cumulative value of all enabled options. The following list describes the value and the corresponding option that is enabled:

Value	Option Enabled
`1`	The KeepIdentity option preserves identity values. If unspecified, identity values are ignored in the source and are assigned by the destination.
`2`	The TableLock option assigns a table lock for the duration of the bulk copy operation. Other applications cannot update the table until the operation completes. If unspecified, the default bulk locking mechanism specified by the database server is used.
`16`	The CheckConstraints option checks integrity constraints while data is being copied. If unspecified, constraints are not checked.
`32`	The FireTriggers option causes the database server to fire insert triggers for the rows being inserted into the database. If unspecified, triggers are not fired.
`64`	The KeepNulls option preserves null values in the destination table regardless of the settings for default values. If unspecified, null values are replaced by column default values where applicable.

Behavior

If set to `0`, all the options are disabled.

Example

A value of `67` means the KeepIdentity, TableLock, and KeepNulls options are enabled (1 + 2 + 64).

Data source method

```
public Long getBulkLoadOptions()
```

```
public void setBulkLoadOptions(Long)
```

Default

2

Data type

Long

See also

- DataDirect Bulk Load on page 94
- Connection property descriptions on page 103

CatalogOptions

Purpose

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

Valid values

`0` | `2`

Behavior

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

If set to `2`, result sets contain synonyms that are returned from the following DatabaseMetaData methods: getFunctions(), getTables(), getColumns(), getProcedures(), getProcedureColumns(), and getFunctionColumns()

Data source method

```
public Integer getCatalogOptions()
```

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

Default

`0`

Data type

Int

ClientHostName

Purpose

The host name, or workstation ID, of the client machine to be stored in the database. This property sets the hostname column of the sysprocesses table in the database.

Valid values

`string`

where:

`string`

is the host name of the client machine.

Notes

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

Data source method

```
public String getClientHostName()
```

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

Default

Empty string

Data type

String

See also

-

- **Note:** For more information on connection client information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference*.

- Connection property descriptions on page 103

ClientUser

Purpose

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

Valid values

string

where:

string

is a valid user ID.

Data source method

```
public String getClientUser()
```

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

Default

Empty string

Data type

String

See also

-

- **Note:** For more information on connection client information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference*.

- Connection property descriptions on page 103

CodePageOverride

Purpose

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

Valid values

string

where:

string

is the name of a valid code page that is supported by your JVM. For example, `CP950`.

Notes

- 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.

Data Source Methods

```
public String getCodePageOverride()
```

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

Default

No default value

Data type

String

ColumnEncryption

Purpose

Specifies whether the driver is enabled for Always Encrypted functionality when accessing data from encrypted columns.

Valid Values

`Disabled` | `ResultsetOnly` | `Enabled`

Behavior

If set to `Disabled`, the driver does not use Always Encrypted functionality. The driver does not attempt to decrypt data from encrypted columns, but will return data as binary formatted cipher text. However, statements containing parameters that reference encrypted columns are not supported and will return an error.

If set to `ResultsetOnly`, the driver transparently decrypts result sets and returns them to the application. Queries containing parameters that affect encrypted columns will return an error.

If set to `Enabled`, the driver fully supports Always Encrypted functionality. The driver transparently decrypts result sets and returns them to the application. In addition, the driver transparently encrypts parameter values that are associated with encrypted columns.

Notes

- When Always Encrypted functionality is enabled, values must be provided for the following properties according to your keystore provider:
  - For Azure Key Vault, you must specify values for the AEKeystorePrincipalId and AEKeystoreClientSecret properties.
  - For Java KeyStore, you must specify values for the AEKeystoreLocation and AEKeystoreSecret properties.
- When Always Encrypted functionality is enabled, the driver transparently supports both randomized encryption and deterministic encryption.
- Parameter markers must be used when specifying values that are associated with encrypted columns. If literal values are specified in a statement targeting encrypted columns, the driver will return an error.

Data source method

```
public String getColumnEncryption()
```

```
public void setColumnEncryption(String)
```

Default

`Disabled`

Data Type

String

See Also

- AEKeystorePrincipalId on page 120

- AEKeystoreClientSecret on page 118
- AEKeystoreLocation on page 119
- AEKeystoreSecret on page 121
- Always Encrypted on page 64
- Performance considerations on page 77

ConnectionRetryCount

Purpose

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.

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.
- If MultiSubnetFailover is enabled and the connection attempt fails, the driver will attempt to connect two more times, regardless of the ConnectionRetryCount setting.

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.

Data source method

```
public Integer getConnectionRetryCount()
```

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

Default

5

Data type

Int

See also

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*.

ConnectionRetryDelay

Purpose

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

Valid values

`0` | *x*

where:

x

is a number of seconds.

Behavior

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

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

Example

If ConnectionRetryCount is set to `2`, 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.

Notes

When MultiSubnetFailover is enabled, the ConnectionRetryDelay connection property is ignored.

Data source method

```
public Integer getConnectionRetryDelay()
```

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

Default

`1` (second)

Data type

Int

See also

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*.

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 this is 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 method

```
public Integer getConvertNull()
```

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

Default

`1`

Data type

Int

See also

Connection property descriptions on page 103

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 | TLSv1.1 | TLSv1 | SSLv3 | SSLv2

Caution: To avoid vulnerabilities associated with SSLv3 and SSLv2, good security practices recommend using TLSv1 or higher.

Example

If your server supports TLSv1.1 and TLSv1.2, you can specify acceptable cryptographic protocols with the following key-value pair:

```
CryptoProtocolVersion=TLSv1.1,TLSv1.2
```

Notes

- The TLSv1.3 protocol can be used for encryption only when the EncryptionMethod property is set to Strict (EncryptionMethod=Strict).
- 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 method

```
public String getCryptoProtocolVersion()
```

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

Default

None

Data type

String

See also

- EncryptionMethod on page 144
- TLS/SSL encryption on page 62

DatabaseName

Purpose

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

Valid Values

string

where:

string

is the name of a SQL Server or Azure database.

Notes

Database can be used as an alias for DatabaseName.

Data source method

`public String getDatabaseName()`

`public void setDatabaseName(String)`

Default

None

Data type

String

DateTimeInputParameterType

Purpose

Specifies how the driver describes the data type for Date/Time/Timestamp input parameters.

This property only applies to connections to Azure and SQL Server 2008 and higher. For prior versions of Microsoft SQL Server, the driver always describes Date/Time/Timestamp input parameters as datetime.

Valid values

`auto` | `dateTime` | `dateTimeOffset`

Behavior

If set to `auto`, the driver uses the following rules to describe the data type of Date/Time/Timestamp input parameters:

- If an input parameter is set using setDate(), the driver describes it as date.
- If an input parameter is set using setTime(), the driver describes it as time.
- If an input parameter is set using setTimestamp(), the driver describes it as datetimeoffset.

If set to `dateTime`, the driver describes Date/Time/Timestamp input parameters as datetime.

If set to `dateTimeOffset`, the driver describes Date/Time/Timestamp input parameters as datetimeoffset.

Data source method

```
public String getDateTimeInputParameterType()
```

```
public void setDateTimeInputParameterType(String)
```

Default

`auto`

Data type

String

See also

Connection property descriptions on page 103

DateTimeOutputParameterType

Purpose

Specifies how the driver describes the data type for Date/Time/Timestamp output parameters.

This property only applies to connections to Microsoft SQL Server 2008 and higher and Microsoft Windows Azure SQL Database. For connections to prior versions of Microsoft SQL Server, the driver always describes Date/Time/Timestamp output parameters as datetime.

Valid values

`auto` | `dateTime` | `dateTimeOffset`

Behavior

If set to `auto`, the driver uses the following rules to describe the data type of Date/Time/Timestamp output parameters:

- If an output parameter is set using setDate(), the driver describes it as date.
- If an output parameter is set using setTime(), the driver describes it as time.
- If an output parameter is set using setTimestamp(), the driver describes it as datetimeoffset.

If set to `dateTime`, the driver describes Date/Time/Timestamp output parameters as datetime.

If set to `dateTimeOffset`, the driver describes Date/Time/Timestamp output parameters as datetimeoffset.

Data source method

```
public String getDateTimeOutputParameterType()
public void setDateTimeOutputParameterType(String)
```

Default

`auto`

Data type

String

See also

Connection property descriptions on page 103

DescribeInputParameters

Purpose

Determines whether the driver attempts to determine, at execute time, which data type to use to send input parameters to the database server. Sending parameters as the data type the database expects improves performance and prevents locking issues caused by data type mismatches.

Valid values

`noDescribe` | `describeIfString` | `describeIfDateTime` | `describeAll`

Behavior

If set to `noDescribe`, the driver does not attempt to describe input parameters and sends String and Date/Time/Timestamp input parameters to the server as specified by the StringInputParameterType and DateTimeInputParameterType properties.

If set to `describeIfString`, the driver submits a request to the database to describe String input parameters. The driver uses the data types that are returned by the driver to determine whether to describe the String input parameters as nvarchar or varchar. If this operation fails, the driver sends String input parameters to the server as specified by the StringInputParameterType property.

If set to `describeIfDateTime`, the driver submits a request to the database to describe Date/Time/Timestamp input parameters. The driver uses the data types that are returned by the driver to determine how to describe the Date/Time/Timestamp input parameters. If this operation fails, the driver sends Date/Time/Timestamp input parameters to the server as specified by the DateTimeInputParameterType property.

If set to `describeAll`, the driver submits a request to the database to describe both String and Date/Time/Timestamp input parameters and uses the data types that are returned by the driver to determine which data type to use to describe the input parameters. If this operation fails, the driver sends String input parameters to the server as specified by the StringInputParameterType property and sends Date/Time/Timestamp input parameters to the server as specified by the DateTimeInputParameterType property.

Data source method

```
public String getDescribeInputParameters()
```

```
public void setDescribeInputParameters(String)
```

Default

noDescribe

Data type

String

See also

Connection property descriptions on page 103

DescribeOutputParameters

Purpose

Determines whether the driver attempts to determine, at execute time, which data type to use to send output parameters to the database server. Sending parameters as the data type the database expects improves performance and prevents locking issues caused by data type mismatches.

Valid values

noDescribe | describeIfString | describeIfDateTime | describeAll

Behavior

If set to noDescribe, the driver does not attempt to describe output parameters and sends String and Date/Time/Timestamp output parameters to the server as specified by the StringOutputParameterType and DateTimeOutputParameterType properties.

If set to describeIfString, the driver submits a request to the database to describe String output parameters. The driver uses the data types that are returned by the driver to determine whether to describe the String output parameters as NVARCHAR or VARCHAR. If this operation fails, the driver sends String output parameters to the server as specified by the StringOutputParameterType property.

If set to describeIfDateTime, the driver submits a request to the database to describe Date/Time/Timestamp output parameters. The driver uses the data types that are returned by the driver to determine how to describe the Date/Time/Timestamp output parameters. If this operation fails, the driver sends Date/Time/Timestamp output parameters to the server as specified by the DateTimeOutputParameterType property.

If set to describeAll, the driver submits a request to the database to describe both String and Date/Time/Timestamp output parameters and uses the data types that are returned by the driver to determine which data type to use to describe the output parameters. If this operation fails, the driver sends String output parameters to the server as specified by the StringOutputParameterType property and sends Date/Time/Timestamp output parameters to the server as specified by the DateTimeOutputParameterType property.

Data source method

```
public String getDescribeOutputParameters()
```

```
public void setDescribeOutputParameters(String)
```

Default

noDescribe

Data type

String

See also

Connection property descriptions on page 103

Domain

Purpose

Specifies the name of the domain server that administers the database. Set this property only if you are using NTLM authentication. If the Domain property is unspecified, the driver tries to determine the domain server name from the User property.

Valid values

string

where:

string

is the name of the domain server.

Data source method

```
public String getDomain()
public void setDomain(String)
```

Default

None

Data type

String

See also

- NTLM authentication on page 60
- AuthenticationMethod on page 125

EnableBulkLoad

Purpose

Specifies whether the driver uses the native bulk load protocols in the database. Bulk load bypasses the data parsing that is usually done by the database, providing an additional performance gain over batch operations. This property allows existing applications with batch inserts to take advantage of bulk load without requiring changes to the application code.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses the database bulk load protocol when an application executes an INSERT with multiple rows of parameter data. If the protocol cannot be used, the driver returns a warning.

If set to `false`, the driver uses standard parameter arrays.

Data source method

```
public Boolean getEnableBulkLoad()
```

```
public void setEnableBulkLoad(Boolean)
```

Default

`false`

Data type

Boolean

See also

- DataDirect Bulk Load on page 94
- Connection property descriptions on page 103
- Performance considerations on page 77

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`

Behavior

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 method

```
public Boolean getEnableCancelTimeout()
```

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

Default

`false`

Data type

Boolean

See also

- Driver specifications on page 32
- Connection property descriptions on page 103

EnableReplicationUser

Purpose

Specifies whether explicit values may be inserted into IDENTITY columns defined as NOT FOR REPLICATION. This property is useful when publishing data from earlier versions of SQL Server.

Valid Values

`true` | `false`

Behavior

If set to `true` (Enabled), the driver allows explicit inserts on IDENTITY columns defined as NOT FOR REPLICATION.

If set to `false` (Disabled), the driver enforces constraints on IDENTITY columns imposed by the NOT FOR REPLICATION flag.

Data source method

```
public Boolean getEnableReplicationUser()
```

```
public void setEnableReplicationUser(Boolean)
```

Default

`false` (Disabled)

Data type

Boolean

See also

- Inserts into IDENTITY columns for data replication on page 102

EncryptionMethod

Purpose

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

Valid values

`noEncryption` | `SSL` | `Strict` | `requestSSL` | `loginSSL`

Behavior

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

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

If set to `Strict`, the driver uses the TDS (Tabular Data Stream) 8.0 protocol to support TLSv1.3 encryption for SQL Server connections. You must specify this value when your server is configured with `Force Strict Encryption=yes`.

Important: When using strict connection encryption:

- The driver validates the certificates sent by the server (`ValidateServerCertificate=true`) for the connection, regardless of the setting of the ValidateServerCertificate property.
- You must specify a truststore containing the server certificate against which the server will be validated at connection.

If set to `requestSSL`, the login request and data is encrypted using TLS/SSL. If the database server does not support TLS/SSL, the driver establishes an unencrypted connection.

If set to `loginSSL`, the login request is encrypted using TLS/SSL. Data is encrypted using TLS/SSL If the database server is configured to require TLS/SSL. If the database server does not require TLS/SSL, data is not encrypted and only the login request is encrypted.

Notes

- For all the values, the TLS/SSL protocol used is determined by the setting of the CryptoProtocolVersion connection property.

- When establishing a connection to Microsoft Azure Synapse Analytics, Microsoft Analytics Platform System, or Microsoft Windows Azure SQL Database, the driver will enable TLS/SSL data encryption by default (`EncryptionMethod=SSL`).
- 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.
- If TLS/SSL is enabled, the driver communicates with database protocol packets that are set by the server's default packet size. Any value set by the PacketSize property is ignored.
- If TLS/SSL is enabled, the following properties also apply:

  CryptoProtocolVersion

  HostNameInCertificate

  TrustStore

  TrustStorePassword

  ValidateServerCertificate

Data source method

```
public String getEncryptionMethod()
```

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

Default

`noEncryption`

Data type

String

See also

- TLS/SSL encryption on page 62
- Connection property descriptions on page 103

FailoverGranularity

Purpose

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

Valid values

`nonAtomic` | `atomic` | `atomicWithRepositioning`

Behavior

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

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

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

Data source method

```
public String getFailoverGranularity()
```

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

Default

`nonAtomic`

Data type

String

See also

- FailoverMode on page 146
-

- **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*.

FailoverMode

Purpose

Specifies the type of failover method the driver uses.

Valid values

`connect` | `extended` | `select`

Behavior

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

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

If set to `select`, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work 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.
- When MultiSubnetFailover is enabled, the driver does not support FailoverMode=`select`. If MultiSubnetFailover=`true` and FailoverMode=`select`, the driver downgrades the FailoverMode to `extended` and provides the following warning.

```
failoverMode=select is not supported for AlwaysOn Availability Group,
downgraded to failoverMode=extended
```

Data source method

```
public String getFailoverMode()
```

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

Default

`connect`

Data type

String

See also

-

- **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*.

- Connection property descriptions on page 103

FailoverPreconnect

Purpose

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

Valid values

`true` | `false`

Behavior

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

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

Notes

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

Data source method

```
public Boolean getFailoverPreconnect()
```

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

Default

`false`

Data type

Boolean

See also

- FailoverMode on page 146
-

- **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*.

FetchTSWTZAsTimestamp

Purpose

Determines whether column values with the datetimeoffset data type are returned as a JDBC VARCHAR or TIMESTAMP data type.

This property only applies to connections to Azure and SQL Server 2008 and higher.

Valid values

`true` | `false`

Behavior

If set to `true`, column values with the datetimeoffset data type are returned as a JDBC TIMESTAMP data type.

If set to `false`, column values with the datetimeoffset data type are returned as a JDBC VARCHAR data type.

Data source method

```
public Boolean getFetchTSWTZAsTimestamp()
```

```
public void setFetchTSWTZAsTimestamp(Boolean)
```

Default

`false`

Data type

Boolean

See also

Connection property descriptions on page 103

FetchTWFSasTime

Purpose

Determines whether the driver returns column values for the native TIME data type as the JDBC TIME or TIMESTAMP data type.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver returns column values for the native TIME data type as the JDBC TIME data type. The fractional seconds portion of the value is truncated when the value is returned in the java.sql.Time object.

If set to `false`, the driver returns column values for the native TIME data type as the JDBC TIMESTAMP data type. The Java Epoch (Jan 1,1970) is returned in the date portion.

Data source method

```
public Boolean getFetchTWFSasTime()
```

```
public void setFetchTWFSasTime(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Connection property descriptions on page 103

GSSCredential

Purpose

Specifies the GSS credential object used to instantiate Kerberos constrained delegation. Constrained delegation is a Kerberos mechanism that allows a client application to delegate authentication to a second service.

Important: Because the value of this property is a Java object, it cannot be specified in a connection URL. It can only be passed as a `Properties` or `DataSource` object.

Valid Values

`string`

where:

`string`

is the name of the GSS credential object.

Notes

- AuthenticationMethod must be set to `kerberos` to use constrained delegation.

Data Source Method

```
public String getGSSCredential()
```

```
public void setGSSCredential(String)
```

Default

Null

Data type

String

See also

- Authentication on page 48
- Kerberos authentication on page 54
- Constrained delegation on page 58

HostNameInCertificate

Purpose

Specifies a host name for certificate validation when 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`

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.

Notes

- If the HostNameInCertificate is not specified, the driver automatically uses the value of the ServerName from the URL as the value for validating the certificate.
- If SSL encryption or certificate validation is not enabled, this property is ignored.
- If SSL encryption and validation is enabled and this property is unspecified, the driver uses the server name that is specified in the connection URL or data source of the connection to validate the certificate.

Data source method

```
public String getHostNameInCertificate()
```

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

Default

Empty string

Data type

String

See also

- EncryptionMethod on page 144
- ValidateServerCertificate on page 184

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 method

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

Default

Empty string

Data type

String

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- Performance considerations on page 77

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

string

where:

string

is one or multiple SQL commands.

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 sets the handling of null values to the Microsoft SQL Server default and allows delimited identifiers:

```
jdbc:datadirect:sqlserver://server1:1433;InitializationString=
(set ANSI_NULLS off;set QUOTED_IDENTIFIER on);DatabaseName=test
```

Data source method

`public String getInitializationString()`

`public void setInitializationString(String)`

Default

None

Data type

String

InsensitiveResultSetBufferSize

Purpose

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

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 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 GB. 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*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, 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 method

```
public Integer getInsensitiveResultSetBufferSize()
```

```
public void setInsensitiveResultSetBufferSize(Integer)
```

Default

`2048` (KB)

Data type

Int

See also

Performance considerations on page 77

JavaDoubleToString

Purpose

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

Valid values

`true` | `false`

Behavior

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

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

Data source method

```
public Boolean getJavaDoubleToString()
```

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

Default

`false`

Data type

Boolean

See also

Connection property descriptions on page 103

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. Additionally, the PROCEDURE_NAME column contains procedure name qualifiers. For example, for the fully qualified procedure named 1.sp_productadd, the driver would return sp_productadd;1.

Data source method

```
public Integer getJDBCBehavior()
```

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

Default

`1`

Data type

Int

See also

Connection property descriptions on page 103

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 method

```
public Boolean getLoadBalancing()
```

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

Default

`false`

Data type

Boolean

See also

- AlternateServers on page 122
-

- **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*.

LoginConfigName

Purpose

Specifies the name of the entry in the JAAS login configuration file that contains the authentication technology used by the driver to establish a Kerberos connection. The LoginModule-specific items found in the entry are passed on to the LoginModule.

Valid values

`entry_name`

where:

`entry_name`

is the name of the entry that contains the authentication technology used with the driver.

Example

In the following example, JDBC_DRIVER_01 is the entry name while the authentication technology and related settings are found in the brackets.

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

Data Source Method

```
public String getLoginConfigName()
```

```
public void setLoginConfigName(String)
```

Default

JDBC_DRIVER_01

Data type

String

See also

- Authentication on page 48
- Kerberos authentication on page 54
- The JAAS login configuration file on page 57

LoginTimeout

Purpose

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

Valid values

0 | *x*

where:

x

is 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.

Notes

When MultiSubnetFailover is enabled, the value of the LoginTimeout property is 15 seconds by default. When LoginTimeout is set to 0 (zero), the driver will still timeout requests after 15 seconds. However, when the value of LoginTimeout is an integer greater than 0 (zero), the driver will timeout requests for the specified duration.

Data source method

```
public Integer getLoginTimeout()
```

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

Default

`0`

Data type

Int

See also

- Driver specifications on page 32
- Connection property descriptions on page 103

LongDataCacheSize

Purpose

Determines whether the driver caches long data (images, pictures, long text, binary data, 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 in KB 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 method

```
public Integer getLongDataCacheSize()
```

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

Default

`2048` (KB)

Data type

Int

See also

Performance considerations on page 77

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

- MaxStatements can be used as an alias for MaxPooledStatements.
- 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 method

```
public Integer getMaxPooledStatements()
```

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

Default

`0`

Data type

Int

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- Performance considerations on page 77
- RegisterStatementPoolMonitorMBean on page 169

MultiSubnetFailover

Purpose

Determines whether the driver attempts parallel connections to the failover IP addresses of an Availability Group during initial connection or a multi-subnet failover. When MultiSubnetFailover is enabled, the driver simultaneously attempts to connect to all IP addresses associated with the Availability Group listener when establishing an initial connection or reconnecting after a connection is broken or the listener IP address becomes unavailable. The first IP address to successfully respond to the request is used for the connection. Using parallel-connection attempts offers improved response time over traditional failover, which attempts to connect to alternate servers one at a time.

Valid values

`true` | `false`

Behavior

If set to `true`, the driver attempts parallel connections to all failover IP addresses in an Availability Group when establishing an initial connection or reconnecting after a connection is broken or the listener IP address becomes unavailable. The first IP address to successfully respond to the request is used for the connection. This setting is only supported when your environment is configured for Always On Availability Groups.

If set to `false`, the driver connects to an alternate server or servers as specified by the AlternateServer property when the primary server is unavailable. Use this setting if your environment is not configured for Always On Availability Groups.

Notes

- When MultiSubnetFalover is enabled, the virtual network name (VNN) of the availability group listener must be specified with the ServerName connection property.
- When MultiSubnetFailover is enabled, the driver does not support FailoverMode=`select`. If MultiSubnetFailover=`true` and FailoverMode=`select`, the driver downgrades the FailoverMode to `extended` and provides the following warning.

  ```
  failoverMode=select is not supported for AlwaysOn Availability Group,
  downgraded to failoverMode=extended
  ```

- When MultiSubnetFailover is enabled, the ConnectionRetryDelay connection property is ignored.
- If MultiSubnetFailover is enabled and the connection attempt fails, the driver will attempt to connect two more times, regardless of the ConnectionRetryCount setting.
- When MultiSubnetFailover is enabled, the value of the LoginTimeout property is 15 seconds by default. When LoginTimeout is set to 0 (zero), the driver will still timeout requests after 15 seconds. However, when the value of LoginTimeout is an integer greater than 0 (zero), the driver will timeout requests for the specified duration.

Data source method

```
public Boolean getMultiSubnetFailover()
```

```
public void setMultiSubnetFailover(Boolean)
```

Default

`false`

Data type

Boolean

See also

- Always On Availability Groups on page 77
-

- **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*.

NetAddress

Purpose

The Media Access Control (MAC) address of the network interface card of the application connecting to Microsoft SQL Server. This value is stored in the net_address column of the sys.sysprocesses table.

Valid values

`string`

where:

string

is a maximum of 12 alphanumeric characters.

Data source method

```
public String getNetAddress()
public void setNetAddress(String)
```

Default

```
000000000000
```

Data type

String

PacketSize

Purpose

Determines the number of bytes for each database protocol packet that is transferred from the database server to the client machine (Microsoft SQL Server refers to this packet as a network packet).

Adjusting the packet size can improve performance. The optimal value depends on the typical size of data that is inserted, updated, or returned by the application and the environment in which it is running. Typically, larger packet sizes work better for large amounts of data. For example, if an application regularly returns character values that are 10,000 characters in length, using a value of 32 (16 KB) typically results in improved performance.

Valid values

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

where:

x

is an integer from 1 to 128 that represents a number of bytes.

Behavior

If set to `-1`, the driver uses the maximum packet size that the database server accepts.

If set to `0`, the driver uses the default packet size configured on the database server.

If set to *x*, the driver uses a packet size that is calculated using the specified value multiplied by 512.

Notes

- If SSL encryption is enabled using the EncryptionMethod property, any value set for the PacketSize property is ignored.
- If your application sends queries that only retrieve small result sets, you may want to use a packet size that is smaller than the maximum packet size that is configured on the database server. If a result set that

contains only one or two rows of data does not completely fill a larger packet, performance will not improve by setting the value to the maximum packet size.

Example

If PacketSize=8, the packet size is set to 8 * 512 bytes (4096 bytes).

Data source method

```
public Integer getPacketSize()
public void setPacketSize(Integer)
```

Default

```
-1
```

Data type

Int

See also

Performance considerations on page 77

Password

Purpose

Specifies a password that is used to connect to the database or instance.

Valid values

string

where

string

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

Data source method

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

Default

None

Data type

String

See also

- Authentication on page 48
- User on page 182

See also

User on page 182

PortNumber

Purpose

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

Valid values

port

where:

port

is the port number.

Data source method

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

Default

1433

Data type

Int

ProgramID

Purpose

The driver name and version information on the client to be stored in the database. This property sets the hostprocess column in the sysprocesses table.

Valid values

string

where:

string

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

Example

576383

Data source method

```
public String getProgramID()
```

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

Default

Empty string

Data type

String

See also

-

- **Note:** For more information on connection client information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference*.

- Connection property descriptions on page 103

ProxyHost

Purpose

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

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

IP_address

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

Data Source Methods

```
public String getProxyHost()
```

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

Default Value

No default value

Data Type

String

See also

ProxyPassword on page 166
ProxyPort on page 167
ProxyUser on page 167

ProxyPassword

Purpose

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

Valid Values

password

where:

password

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

Data Source Methods

```
public String getProxyPassword()
```

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

Default Value

No default value

Data Type

String

See also

ProxyUser on page 167
ProxyPort on page 167
ProxyHost on page 165

ProxyPort

Purpose

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

Valid Values

port

where:

port

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

Data Source Methods

```
public Integer getProxyPort()
```

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

Default Value

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

For HTTP: `80`

For HTTPS: `443`

Data Type

Integer

See also

ProxyUser on page 167
ProxyPassword on page 166
ProxyHost on page 165

ProxyUser

Purpose

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

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Data Source Methods

```
public String getProxyUser()
```

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

Default Value

No default value

Data Type

String

See also

ProxyHost on page 165
ProxyPassword on page 166
ProxyPort on page 167

QueryTimeout

Purpose

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

Valid values

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

where:

x

is a number of seconds.

Behavior

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

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

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

Data source method

```
public Integer getQueryTimeout()
```

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

Default

`0`

Data type

Int

See also

- Driver specifications on page 32
- Connection property descriptions on page 103

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 method

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

Default

`false`

Data type

Boolean

See also

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

- MaxPooledStatements on page 159

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 method

```
public Integer getResultSetMetaDataOptions()
```

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

Default

`0`

Data type

Int

See also

Performance considerations on page 77

SelectMethod

Purpose

A hint to the driver that determines whether the driver requests a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method.

Valid values

`direct | cursor`

Behavior

If set to `direct`, the database server sends the complete result set in a single response to the driver when responding to a query. A server-side database cursor is not created if the requested result set type is a forward-only result set. Typically, responses are not cached by the driver. Using this method, the driver must process the entire response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Typically, the direct method performs better than the cursor method.

If set to `cursor`, a server-side cursor is requested. When returning forward-only result sets, the rows are returned from the server in blocks. The setFetchSize() method can be used to control the number of rows that are returned for each request when forward-only result sets are returned. Performance tests show that, when returning forward-only result sets, the value of Statement.setFetchSize() significantly impacts performance. There is no simple rule for determining the setFetchSize() value that you should use. We recommend that you experiment with different setFetchSize() values to determine which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used.

Notes

SelectMethod=`cursor` is not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System. For these environments, the database server sends the complete result set in a single response to the driver when responding to a query, and a server-side cursor is not created.

Data source method

```
public String getSelectMethod()
```

```
public void setSelectMethod(String)
```

Default

`direct`

Data type

String

See also

Performance considerations on page 77

ServerName

Purpose

Specifies the name or IP address of the server to which you want to connect.

Valid values

`IP_address` | *`named_server`* | *`named_instance`* | *`virtual_network_name`*

where:

IP_address

is the IP address of the server to which you want to connect. For example, you can enter `199.226.224.34`. The IP address can be specified in either IPv4 or IPv6 format, or a combination of the two. See IP addresses on page 73 for details about these formats.

named_server

is the named server address of the server to which you want to connect. For example, you can enter `MyServer`.

named_instance

is a named instance of Microsoft SQL Server. The named instance should be specified as *server_name*`\\`*instance_name*, where *server_name* is the IP address and *instance_name* is the name of the instance to which you want to connect on the specified server. For example, `server1\\instance1`.

virtual_network_name

is the virtual network name (VNN) of the availability group listener when using an Always On Availability Group.

Notes

- For Microsoft Azure Synapse Analytics and Microsoft Analytics Platform System (APS), specifying an IP address for the server is not supported. You must provide a named server to connect.

Data source method

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

Default

None

Data type

String

See also

Connecting to named instances on page 68

ServicePrincipalName

Purpose

Specifies the service principal name to be used for Kerberos authentication.

Valid Values

ServicePrincipalName

where:

ServicePrincipalName

is the four-part service principal name registered with the key distribution center (KDC).

Specify the service principal name using the following format.

Service_Name/Fully_Qualified_Domain_Name:*Port_Number*@*REALM*.COM

where:

Service_Name

is the name of the service hosting the instance. The *Service_Name* for Microsoft SQL Server is MSSQLSvc.

Fully_Qualified_Domain_Name

is the fully qualified domain name (FQDN) of the host machine. This value must match the FQDN registered with the KDC. The FQDN consists of a host name and a domain name. For the example myserver.example.com, myserver is the host name and example.com is the domain name.

Port_Number

is the port number as specified by the PortNumber property.

REALM.COM

is the domain name of the host machine. This value is optional. If no value is specified, the default domain is used. The domain must specified in upper-case characters. For example, EXAMPLE.COM. For Windows Entra ID, the Kerberos realm name is the Windows domain name.

Example

The following is an example of a valid service principal name:

```
MSSQLSvc/myserver.example.com:1433@EXAMPLE.COM
```

Notes

- The driver builds a service principal name in the following manner.
  - MSSQLSvc is used as the service name.
  - The value of the ServerName property is used as the FQDN.
  - The PortNumber property specifies the port number that is used.
  - The default realm in the krb5.conf file is used as the realm name.
- If the default does not match the service principal name registered with the KDC, then you can specify the value of the service principal name registered with the KDC.
- In a Kerberos configuration, an IP address cannot be used as a FQDN.

Data Source Method

```
public String getServicePrincipalName()
public void setServicePrincipalName(String)
```

Default

Driver builds value based on environment

Data type

String

See also

- Authentication on page 48
- AuthenticationMethod on page 125
- Kerberos authentication on page 54

SnapshotSerializable

Purpose

Allows your application to use Snapshot Isolation for connections.

This property is useful for applications that have the Serializable isolation level set. Using the SnapshotSerializable property allows you to use Snapshot Isolation with no or minimum code changes. If you are developing a new application, you may find that using the constant TRANSACTION_SNAPSHOT is a better choice.

Valid values

`true` | `false`

Behavior

If set to `true` and your application has the transaction isolation level set to Serializable, the application uses Snapshot Isolation for connections.

If set to `false` and your application has the transaction isolation level set to Serializable, the application uses the Serializable isolation level.

Notes

To use Snapshot Isolation, your database also must be configured for Snapshot Isolation.

Data source method

```
public Boolean getSnapshotSerializable()
```

```
public void setSnapshotSerializable(Boolean)
```

Default

`false`

Data type

Boolean

See also

- Driver specifications on page 32
- Snapshot isolation level on page 87
- Performance considerations on page 77

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls 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 yes.

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.

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)
```

Data source method

```
public String getSpyAttributes()
```

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

Default

None

Data type

String

See also

Refer to "Tracking JDBC Calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Spy.

StringInputParameterType

Purpose

Determines whether the driver sends String input parameters to the database in Unicode or in the default character encoding of the database.

Valid values

`nvarchar` | `varchar`

Behavior

If set to `nvarchar`, the driver sends String input parameters to the database in Unicode.

If set to `varchar`, the driver sends String input parameters to the database in the default character encoding of the database. This value can improve performance because the server does not need to convert Unicode characters to the default encoding.

Notes

If a value is specified for the CodePageOverride property and this property is set to `nvarchar`, this property is ignored and a warning is generated.

Data source method

```
public String getStringInputParameterType()
```

```
public void setStringInputParameterType(String)
```

Default

`nvarchar`

Data type

String

See also

Performance considerations on page 77

StringOutputParameterType

Purpose

Determines whether the driver sends String output parameters to the database server in Unicode or in the default character encoding of the database.

Valid values

`nvarchar` | `varchar`

Behavior

If set to `nvarchar`, the driver sends String output parameters to the database as nvarchar(4000). Use this value when all output parameters that are returned in the connection are nchar or nvarchar. If the output parameters are char or varchar, the driver returns the output parameter value, but the returned value is limited to 4000 characters.

If set to `varchar`, the driver sends String output parameters to the database as varchar(8000). Use this value if all output parameters that are returned in the connection are char or varchar. If an output parameter is nchar or nvarchar, the value may not be returned correctly (for example, if the returned value uses a code page other than the default encoding).

Data source method

```
public String getStringOutputParameterType()
```

```
public void setStringOutputParameterType(String)
```

Default

`nvarchar`

Data type

String

See also

Performance considerations on page 77

SuppressConnectionWarnings

Purpose

Determines whether the driver suppresses "changed database" and "changed language" warnings when connecting to the database server.

Valid values

`true` | `false`

Behavior

If set to `true`, warnings are suppressed.

If set to `false`, warnings are not suppressed.

Data source method

```
public Boolean getSuppressConnectionWarnings()
```

```
public void setSuppressConnectionWarnings(Boolean)
```

Default

`false`

Data type

Boolean

TransactionMode

Purpose

Controls how the driver delimits the start of a local transaction.

Valid values

`implicit` | `explicit`

Behavior

If set to `implicit`, the driver uses implicit transaction mode. This means that the database, not the driver, automatically starts a transaction when a transactionable statement is executed. Typically, implicit transaction mode is more efficient than explicit transaction mode because the driver does not have to send commands to start a transaction and a transaction is not started until it is needed. When TRUNCATE TABLE statements are used with implicit transaction mode, the database may roll back the transaction if an error occurs. If this occurs, use the explicit value for this property.

If set to `explicit`, the driver uses explicit transaction mode. This means that the driver, not the database starts a new transaction if the previous transaction was committed or rolled back.

Data source method

```
public String getTransactionMode()
```

```
public void setTransactionMode(String)
```

Default

`implicit`

Data type

String

TruncateFractionalSeconds

Purpose

Determines whether the driver truncates timestamp values to three fractional seconds. For example, a value of the datetime2 data type can have a maximum of seven fractional seconds.

Valid values

`true` | `false`

Behavior

If set to `true`, the driver truncates all timestamp values to three fractional seconds.

If set to `false`, the driver does not truncate fractional seconds.

Data source method

```
public Boolean getTruncateFractionalSeconds()
```

```
public void setTruncateFractionalSeconds(Boolean)
```

Default

`true`

Data type

Boolean

TrustStore

Purpose

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

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

This property is ignored if ValidateServerCertificate=`false`.

Valid values

`string`

where:

`string`

is the directory of the truststore file.

Data source method

```
public String getTrustStore()
```

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

Default

None

Data type

String

See also

- EncryptionMethod on page 144
- ValidateServerCertificate on page 184

TrustStorePassword

Purpose

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

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

This property is ignored if ValidateServerCertificate=`false`.

Valid values

`string`

where:

`string`

is a valid password for the truststore file.

Data source method

```
public String getTrustStorePassword()
```

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

Default

None

Data type

String

See also

- EncryptionMethod on page 144
- ValidateServerCertificate on page 184

User

Purpose

Specifies one of the following identifiers used for authentication:

- The default user ID that is used to connect to your database. This value is used when authenticating with user ID/password authentication (`AuthenticationMethod=userIdPassword`).
- The domain server name used for NTLM authentication (`AuthenticationMethod=ntlmjava` or `AuthenticationMethod=ntlm2java`).
- The client ID of the managed identity for a Microsoft Entra ID user-assigned managed identity authentication.

Valid values

`[` *`domain_name`* `\]` *`user_name`*

where:

domain_name

is the name of a valid domain server. The name is case-insensitive and optional. If specified, you must separate the domain name from the user name by a backward slash (\).

user_name

is a valid user name or a client ID of the user-assigned managed identity. It is case-insensitive.

Example

- `DOMAIN1\Smith` for NTLM authentication.
- `Smith` for user ID/password authentication.
- `e4f62g67-5678-4d56-a567-12a3bc456789` for Microsoft Entra ID managed identity authentication.

Data source method

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

Default

None

Data type

String

See also

Password on page 163
Authentication on page 48

UseServerSideUpdatableCursors

Purpose

Determines whether the driver uses server-side cursors when an updatable result set is requested.

Valid values

`true` | `false`

Behavior

If set to `true`, server-side updatable cursors are created when an updatable result set is requested.

If set to `false`, the client-side updatable cursors are created when an updatable result set is requested.

Notes

Server-side updatable cursors are not supported for Microsoft Azure Synapse Analytics or Microsoft Analytics Platform System.

Data source method

```
public Boolean getUseServerSideUpdatableCursors()
public void setUseServerSideUpdatableCursors(Boolean)
```

Default

`false`

Data type

Boolean

See also

- Server-side updatable cursors on page 88
- Performance considerations on page 77

ValidateServerCertificate

Purpose

Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled (`EncryptionMethod=SSL`). When using 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

- This property is ignored when `EncryptionMethod=Strict`. The driver always validates the server certificate when using strict connection encryption.
- Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties.

Data source method

```
public Boolean getValidateServerCertificate()
```

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

Default

```
true
```

Data type

Boolean

See also

- EncryptionMethod on page 144
- HostNameInCertificate on page 151

XATransactionGroup

Purpose

The transaction group ID that identifies any distributed transactions that are initiated by the connection. This ID can be used for distributed transaction cleanup purposes.

You can use the XAResource.recover method to roll back any transactions left in an unprepared state. When you call XAResource.recover, any unprepared transactions that match the ID on the connection used to call XAResource.recover are rolled back.

Valid values

string

where:

string

is a valid transaction group ID.

Example

If you specify `XATransactionGroup=ACCT200` and call XAResource.recover on the same connection, any transactions that are left in an unprepared state identified by the transaction group ID of ACCT200 are rolled back.

Data source method

```
public String getXATransactionGroup()
```

```
public void setXATransactionGroup(String)
```

Default

None

Data type

String

See also

Distributed transaction cleanup on page 90

XMLDescribeType

Purpose

Determines whether the driver maps XML data to the LONGVARCHAR or LONGVARBINARY data type.

Valid values

`longvarchar` | `longvarbinary`

Behavior

If set to `longvarchar`, the driver maps XML data to the LONGVARCHAR data type.

If set to `longvarbinary`, the driver maps XML data to the LONGVARBINARY data type.

Data source method

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

Default

None

Data type

String

See also

- Returning and inserting/updating XML data on page 82
- Connection property descriptions on page 103